Ultrafast Molecular Dynamics of Model Biological Systems

by

Ruth A. Livingstone

Submitted for the degree of Doctor of Philosophy

Heriot Watt University

School of Engineering and Physical Sciences,
Institute of Photonics and Quantum Sciences

31st August 2012

Abstract

Femtosecond time resolved photoelectron imaging spectroscopy equipment was designed, constructed, and used to reveal the non-adiabatic dynamics of model biological systems. Indole and phenol derivatives were studied as models for eumelanin, a pigment found in humans designed to protect the body from ultraviolet radiation. The photo-dynamics of these molecules was studied after excitation with ultraviolet radiation, with particular emphasis on the effect that the hydroxyl groups have on the $\pi\sigma^*$ dissociative state. It was found that adding a hydroxyl group onto indole to create 5-hydroxyindole had little significant effect on the photodynamics at the excitation wavelengths studied. Adding a second hydroxyl group to phenol had a strongly marked effect only when the hydroxyl groups were in close proximity to each other, in which case it dramatically increased the relaxation rate. An ultrafast optical system, imaging photoelectron spectrometer, and software to control the hardware, and collect and analyse photoelectron data were successfully implemented and used to collect and analyse data. This system will be of use for many more years and will be the basis of much future research.

Acknowledgements

Ruth would like to thank: firstly Dave Townsend, for his support, wisdom and enthusiastic guidance. Secondly James Thompson, Oliver Schalk, Grant Paterson, Martin Paterson, Andrey Boguslavskiy, Ross Donaldson, Therese Bergendahl, and Maria Iljina, for their help, encouragement, helpful discussions and contributions towards the work contained in this thesis. Thirdly Heriot Watt University; the Engineering and Physical Sciences Research Council (EPSRC); and Canada/UK University Partnerships Program (CUUPP) for financial and practical support both for me personally and for the experimental equipment, travel and resources required to conduct this project. Last but not least to, Emily and Gordon Ackerman and Andrew Livingstone for their moral support, encouragement, and practical help with this thesis. Without all of you, this wouldn't have been possible.

ACADEMIC REGISTRY
Research Thesis Submission

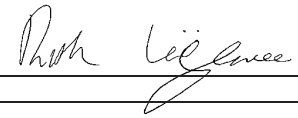
Name*:*	**Ruth Livingstone**		
School/PGI:	**School of Engineering and Physical Sciences**		
Version: *(i.e. First, Resubmission, Final)*	**Final**	Degree Sought (Award **and** Subject area)	**Doctor of Philosophy in Physics**

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate*:*		Date:	

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in the Student Service Centre (SSC)

Received in the SSC by *(name in capitals):*			
Method of Submission *(Handed in to SSC; posted through internal/external mail):*			
E-thesis Submitted (mandatory for final theses)			
Signature:		Date:	

Please note this form should be bound into the submitted thesis.

Updated February 2008, November 2008, February 2009, January 2011

Contents:

1. Introduction

1.1. Ultrafast Molecular Dynamics – Techniques and Relevance

In its long and varied history, chemistry has sought to answer one question above all others. This question relates to how reactants turn into products, (the chemical arrow, Figure 1.1) and still poses a challenge to the modern scientist. Molecules are governed by electrical forces that drive the physical and chemical processes, both internally, such as ionization, excitation and dissociation, and externally through interactions, i.e. solvation, polymerization, and bond formation. Understanding the dynamical evolution of these electrical forces gives us a deeper understanding of the molecular processes and the timescales involved. Molecular dynamics seeks to follow (or infer) the path along the chemical arrow as reactants evolve into products. There are three main experimental methods of deducing molecular dynamics, as will be discussed in more detail later in this chapter. These three methods are: following the disappearance of reactants in 'real time'; following the appearance of products 'in real time'; and probing the asymptotic product state distribution. Within these three methods there are many different techniques and observables that all give complementary information. These techniques, especially when combined with theoretical calculations, can be very powerful tools in providing a deeper understanding of the molecules around us. The development of ultrafast time-resolved methods for probing molecular dynamics, resulted in the 1999 Nobel Prize in Chemistry being awarded to Professor A. Zewail for his role in pioneering the development of this approach [1]. Such time-resolved techniques are central to the work presented in this thesis.

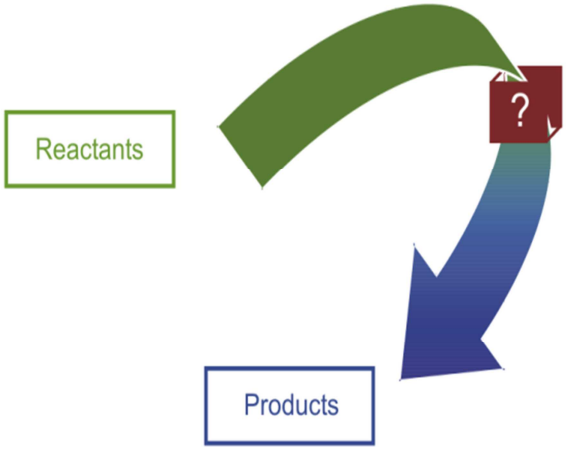


Figure 1.1: The chemical arrow.

One variant of the first main experimental method, femtosecond time-resolved photoelectron spectroscopy [2-5], is explored in this thesis. It has the ability to observe molecules and atoms transitioning on the time scale of interatomic vibrational motion itself, and so can observe these transitions in real time [6]. Modern lasers with the capability of producing pulses from 5 – 100 fs (1 femtosecond = 10^{-15} s) are used to

provide “ultrafast” pulses, which can be used to pump and probe the systems of photochemical interest. For the experiments described in Chapters 2 and 4 of this thesis, an initial “pump” pulse prepares the molecule in an excited electronic state. After a precisely controlled time interval a secondary “probe” pulse ionises the molecule. Examining the ion or electron kinetic energies and angular directions can reveal how the molecule relaxes over time. This technique has proved popular since its development in the 80s [7, 8]. It provides a multiply differential method, which reveals time-, energy-, and angular-resolved information simultaneously. This can reveal deep insights into the dynamics of complex molecules that are otherwise unachievable.

Other techniques such as photoabsorption or photoemission spectroscopy rely on transitions between bound states to probe the electronic state relaxation [9-11]. These events have rigid selection rules, as is discussed in Section 1.2.2. so there are ‘dark’ states that cannot be observed. Photoionisation does not have this problem, as ionisation can happen from any electronic state, subject to Franck-Condon considerations, as discussed in Section 1.2.2. This means that a much fuller dynamical picture can, in principle, be explored [6].

This thesis looks to investigate the photo-dynamics that occur in molecules of biological relevance using time-resolved photoelectron spectroscopy. This chapter puts this work into the context of the wider scientific community by exploring the history and varied techniques used to understand the photo-dynamics of biological molecules. This chapter discusses methods of relevance to our studies, starting by looking at why the molecules that we study are important, and the interesting dynamical processes that can occur within them. Section 1.3 is a general overview of some of the experimental techniques required to investigate the dynamics of these molecules, discussing various forms of spectroscopy, photoelectron imaging techniques, and methods of getting large molecules into the gas phase. Section 1.4 then goes on to look at the optical systems required to carry out the time-resolved excited state spectroscopy that is used in this thesis, including the creation, characterisation, and modification of femtosecond optical pulses.

1.2. Model Biological Systems

1.2.1. Photo-Protection

Since the beginnings of life on Earth, specific molecules have been selected for use as the "building blocks" of life. One fundamental example of this may be found in DNA. If the DNA molecule is damaged this could cause harmful mutations of the information it carries [3, 12]. Another example is the group of biological pigmentation molecules known as melanin, one class of which is eumelanin, found in our hair, skin and eyes. Eumelanin's primary function is to be the first line of defence in protecting us from the potentially damaging effects of ultra-violet (UV) radiation [13]. It is made up of three main building blocks, 5,6-dihydroxyindole (DHI), indolequinone (IQ) and 5,6-dihydroxyindole-2-carboxylic acid (DHICA) that are strikingly similar in shape to the DNA bases adenine and guanine (Figure 1.2).

An important question to ask is what is inherently special about this configuration that it is used in such vital roles across a diverse range of living organisms. One key idea centres on the idea of photostability. In the early years on earth there was no ozone layer to protect organisms from potentially damaging UV radiation. Any excess energy absorbed must always be dispersed within the biological cells. There are various ways a

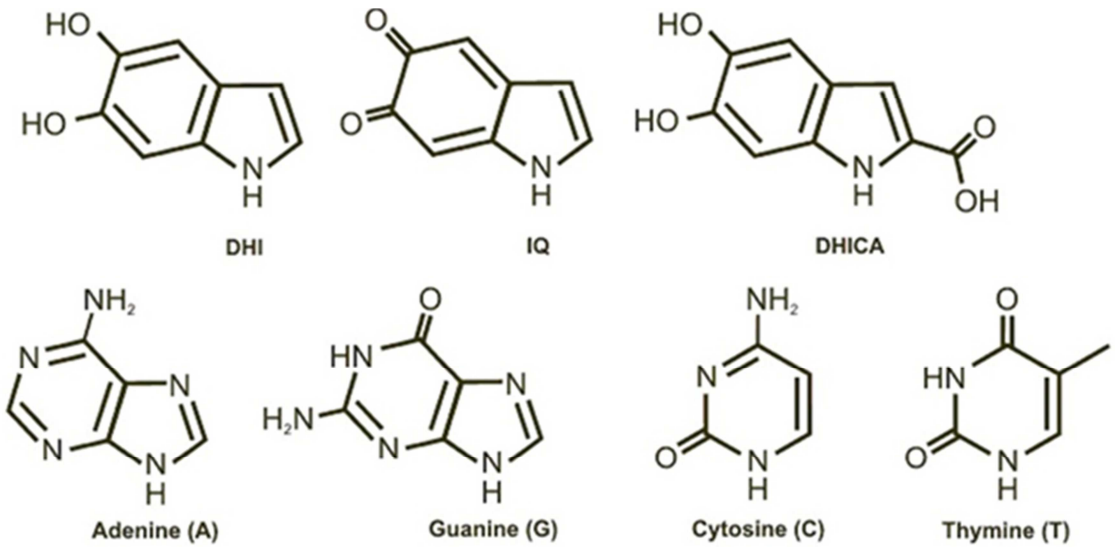


Figure 1.2: Top: The four DNA basis. Bottom: The three constituent building blocks of the eumelanin polymer, 5,6-dihydroxyindole (DHI), indolequinone (IQ) and 5,6-dihydroxyindole-2-carboxylic acid (DHICA).

molecule may do this: a bond in the molecule can be broken, which may have serious implications for a biological framework; the molecule can fluoresce, emitting a photon of light; or the electronic energy can be coupled into vibrational degrees of freedom that can easily dissipate as heat into the surrounding environment. This last case is a breakdown of the Born-Oppenheimer approximation and is known as non-adiabatic dynamics. This will be explained in more detail in the next section. It is of special interest as it is a method of molecular relaxation that can occur on the ultrafast timescale required for these molecules to be photo-stable under ultraviolet illumination. Fluorescence quantum yields have been shown to be very low in the molecules shown in Figure 1.2 [14]. It has been postulated that the non-adiabatic process is particularly rapid and efficient for these molecules, causing higher than normal photostability [15]. If it is found that both DNA bases and eumelanins are photostable then self-protection can be seen at work, with melanins being the first line of defence, and DNA itself being the last line of defence against the harmful effects of UV radiation.

The advance of femtosecond pulsed lasers has made it possible to probe the dynamics of these molecules in real time in a way that was not possible before. In order to discover the photo-chemistry of many molecules, femtosecond lasers can be employed to provide time-resolved ultrafast measurements of the spectroscopy of these molecules [4, 6, 16]. When these measurements are applied to biological molecules, we can hope to gain a better understanding of the way Nature works [12].

Large biological molecules have many states and this creates problems in interpreting experimental results. In order to better understand these molecules, a stepwise approach may be taken [17] by starting with a relatively simple molecule and then systematically incorporating additional substituent groups. Comparing the dynamics of these different systems can yield deeper insights into the role of different substituent groups within a larger molecule. In order to assist in understanding the photoexcitation dynamics within a molecule, theoretical calculations that identify the possible relaxation routes based on molecular orbitals can often be invaluable.

1.2.2. Molecular Orbitals and Non-Adiabatic Dynamics in Biological Molecules

The goal of molecular dynamics is to understand how the electrons and nuclei that make up a molecule move and interact with each other, and how they react to certain events, like excitation or a chemical reaction. In order to understand this we can start by understanding the theoretical framework of how electrons move and interact within molecules. The probability of an electron being found at a certain position relative to a nucleus can be described in terms of an electron orbital. This can be visualised by showing a boundary surface (see Figure 1.3), that captures a percentage of the electron probability, usually 90%. When atoms are part of a molecule, the electron orbitals are no longer affected by a single nucleus. The different atomic orbitals can constructively or destructively interfere with each other to create a set of molecular orbitals. There are three main types of molecular orbitals: bonding orbitals that, if occupied, contribute to the stability of a bond in the molecule; anti-bonding orbitals that decrease the strength of a molecule (often denoted with a * e.g. σ*), and non-bonding orbitals that have no effect on the stability of the molecule as a whole. Molecular orbitals that are centred around the bond axis are generally termed σ-orbitals, while π-orbitals have a nodal plane that passes through the nuclei involved in the bond. Figure 1.3 and Figure 1.12 have some examples of π and σ orbitals. Understanding molecular orbitals is essential to calculate and understand the dynamics of molecules and molecular reactions.

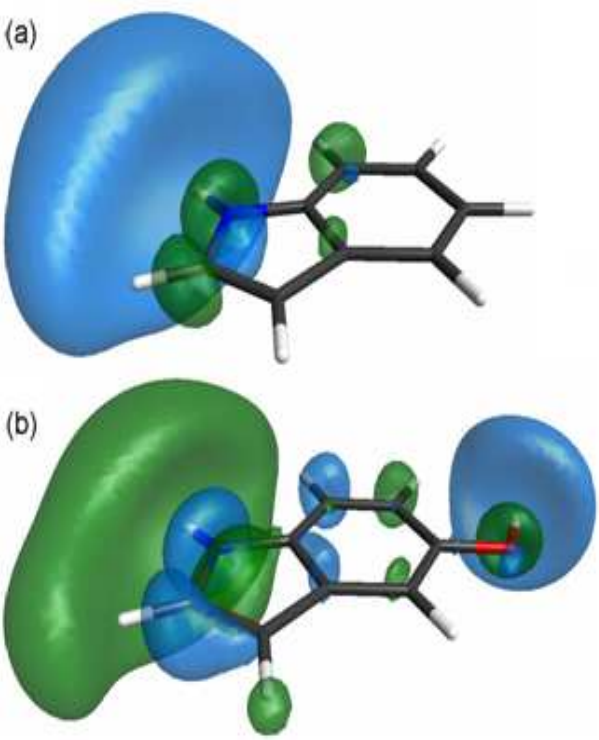


Figure 1.3: Illustration of σ molecular orbitals in (a) indole and (b) 5-hydroxyindole.*

The Born-Oppenheimer approximation [18] relies on the fact that the movement of the electrons are so rapid in comparison to the movements of the nuclei, that the motions are not coupled or related to one another. The electrons see the nuclei as stationary and the nuclei see the electrons as a blurred out average background field.

This is the central theory underpinning all of molecular spectroscopy and dynamics. With this assumption one can take any arbitrary position of the nuclei in a molecule and calculate the molecular orbitals and states at that position. This allows us to draw potential curves and surfaces over which the nuclear (i.e. vibrational and rotational) motion evolves.

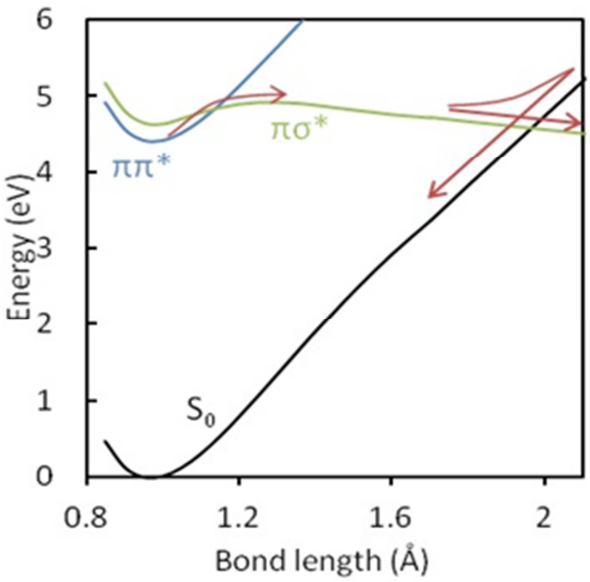


Figure 1.4: Potential energy curves for 3 electronic states, illustrating the role of the πσ state in relaxation of excited hetero-aromatic molecules. This is an unpublished calculation along the N-H bond in indole.*

Electronic energy states within a molecule are defined as the potential energy that a molecule has when its electrons are in a certain set of orbitals. The ground state energy is when all the electrons are in the lowest possible orbitals subject to the population requirements given by Pauli principles. When the molecule is relaxed into the minimum energy position it has a certain energy, we often label this energy as zero, and define all the other energies relative to it. This gives us a measure of how much energy we will need to put into a molecule to move it into a different electronic state. For example, the πσ* state energy is the potential energy of a molecule with an electron in an anti-bonding σ* orbital, and a hole in a usually occupied π orbital. The difference between the ground state energy and the πσ* state energy tells us the amount of energy required to move an electron from the π orbital to the σ* orbital. A potential energy curve or surface (see Figure 1.4) picks one or more co-ordinates along which the molecule can evolve, such as a bond stretching or rotating, and maps what happens to the energy within the molecule as the molecule evolves along these coordinates.

When a molecule moves from one state to another, often by absorbing or emitting a photon, there are certain principles that govern which states a molecule can easily transition to, and which transitions are "forbidden". This is true for electronic, vibrational and rotational states, although rotational state transitions are not discussed in this thesis. The following equation describes the transition dipole moment $\mathbf{R^{nm}}$ for a molecule going from an electronic state, with an electronic wavefunction Ψ''_{Elec} and a vibrational wavefunction Ψ''_{Vib} to a new state with wavefunctions Ψ'_{Elec} and Ψ'_{Vib}.

$$\mathbf{R^{nm}} = \sum_{\varepsilon} (-1)^{\varepsilon-\varepsilon_z} \left[\langle \Psi'_{Elec} | \hat{\mu} \cdot \hat{E} | \Psi''_{Elec} \rangle \langle \Psi'_{Vib} | \Psi''_{Vib} \rangle + \left\langle \Psi'_{Elec} \left| \frac{d(\hat{\mu} \cdot \hat{E})}{dR} \right| \Psi''_{Elec} \right\rangle \langle \Psi'_{Vib} | R | \Psi''_{Vib} \rangle \right] \quad (1.1)$$

The term highlighted in red gives us selection rules about electronic transitions. $\hat{\mu}$ represents the electron dipole moment, and $\hat{E}$ the photon electric field vector for the photon that is either emitted or absorbed in the transmission. The final, green, terms express the dependence of these values on the inter-nuclear coordinate R. Evaluating this integral will give us the strength (or likelihood) of this transition, however because of symmetry considerations this term is often zero, meaning the transition has no probability of occurring.

There are many types of symmetry that a molecule, an orbital, or a vibrational wavefunction may have. An expansive explanation of symmetry is beyond the scope of this thesis; however, it is covered very briefly below. Symmetry operations are actions that you can perform on an object which leave the object indistinguishable from its starting position. Symmetry elements are points of reference about which these operations can take place, and can be a point, line or plane. For example if a molecule has a rotational axis of symmetry, then if you rotate the molecule $\frac{360^o}{n}$ around the axis, it will look exactly the same. This is called an n-fold rotational axis, and in abbreviated $\mathbf{C_n}$. For example $\mathbf{C_4}$ symmetry means that if you rotate a molecule 90° it looks identical to itself. Simple examples of rotational symmetry can be seen in Figure 1.5. Other examples are planes of symmetry (**σ**), inversion centres (**i**), rotation-reflection axes (**S**), and if there is no symmetry at all, identity (**E**). Sets of symmetry operations are called point groups, and a molecule or object can be assigned to a point group if it is left unchanged by every symmetry operation in the group. Various chemical properties, for example the polarity, depend in the symmetry of a molecule. The molecular orbitals or

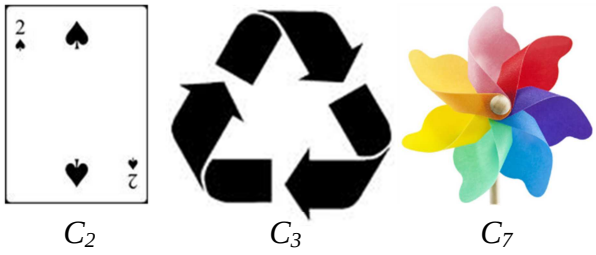


Figure 1.5: Everyday objects demonstrating rotational symmetry.

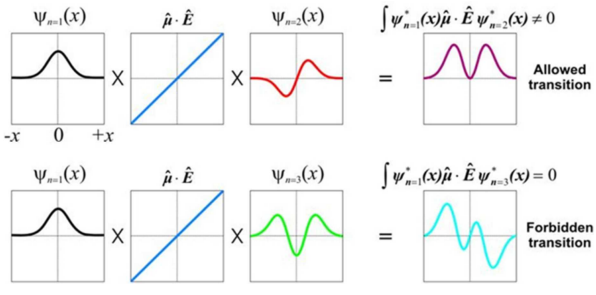


Figure 1.6: Cartoon demonstrating the effect of symmetry on the wavefunction integral – and how this can lead to allowed and forbidden transitions.

wavefunctions may have symmetry that is different from that of the molecular framework. Character tables demonstrate whether the wavefunction is symmetric (1) or anti-symmetric (-1) for each symmetry operation. The simplest way of gaining information about an electronic transition is to evaluate whether or not the transition is allowed, by examining the symmetries of the terms in $\langle \Psi'_{Elec} | \hat{\mu}_{\mathrm{Int}}^{\mathrm{Elec}} | \Psi''_{Elec} \rangle$. If a wavefunction is symmetric, it is given the value 1. If it is anti-symmetric it is given the value -1. The photon is always considered as anti-symmetric. When these values are multiplied, if the outcome is 1, the transition is allowed. If not it is forbidden. This is due to the fact that the integral of an anti-symmetric function is always zero, and so we can quickly identify without an explicit evaluation of the integral whether the outcome will be zero or non-zero. This concept is explored visually in Figure 1.6 If two photons are involved, then different transitions are allowed, as the term in the centre becomes -1×-1=1. These simple rules allow the forbidden electronic transitions to be easily identified.

The blue term in Equation 1.1 gives us selection rules about the transition between the various vibrational levels within the electronic states. The same symmetry considerations can be applied, but we can gain additional knowledge by understanding the wavefunctions themselves. The Franck-Condon principle states that a vibronic transition is more likely to occur if there is significant overlap between the vibrational wavefunctions (see Figure 1.7). The intensity (or likelihood) of a transition is proportional to the square of the overlap integral between the two wavefunctions: $S_{v'v''} \propto \langle \Psi'_{Vib} | \Psi''_{Vib} \rangle^2$ [19]. Figure 1.7 shows the first six vibrational wavefunctions in two electronic states, and highlights two

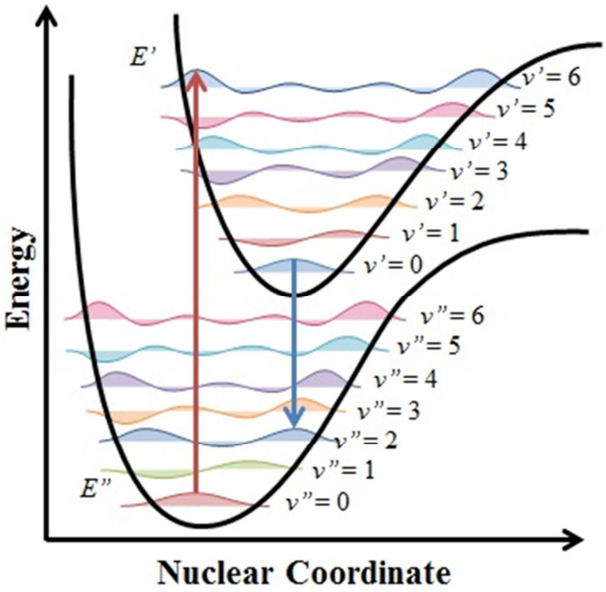


Figure 1.7: Cartoon demonstrating the Franck-Condon Principle. Transitions between levels are favoured when they correspond to a minimal change in the nuclear coordinate. Levels with the greatest overlap of vibrational wavefunction will be most likely to be involved in emission or absorption events.

transitions with a high overlap integral. It can also be seen from this figure that the energy required to transition between the two electronic states E'' and E' is highly dependent on the initial vibrational level. These rules are also important when considering molecular ionisation. If you consider E' as any state in a molecule, and E'' as the ground ionic state in the same molecule, the energy required to ionise the molecule will depend on both the vibrational level of the initial state, and the relative shape and position of the two potential energy surfaces. This means that for a photon with a given energy, some states that would seem to accessible, may actually have a very small probability of ionisation.

In addition to this, Koopman's theorem makes the assumption that when a molecule is ionised, the remaining electrons do not instantaneously rearrange themselves. This is called the frozen-core approximation and means that, apart from the electron that has been removed, all the orbitals that were occupied in the neutral species remain occupied in the cation. The energy required to ionise the molecule is not the difference between the molecule's excited state and the ionic ground state, but the difference between the molecular doubly excited state and the cation singly excited state. In other words, the ionisation energy (I) should be the same as the orbital energy (ε) of the extracted electron. This is stated as the Koopman's theorem [20]:

$$I \approx \varepsilon \tag{1.2}$$

This assumption neglects any effects due to the re-arrangement of electrons that are left behind in the cation; however these effects are usually minimal. Koopman's theory must be taken into account when assigning observed photoelectron kinetic energies to electronic states within neutral molecules. When a molecule is ionised, any excess energy given by the photon, over and above the ionisation energy, is given to the electron and the cation as kinetic energy. According to this assumption, the total kinetic energy (E_k) after an ionisation event is related to the photon energy ($h\nu$) by the following relation:

$$E_k = h\nu - \varepsilon \tag{1.3}$$

Each electronic state correlates - upon removal of the outermost electron - to an electronic state of the cation [21]. This is illustrated in Figure 1.8. If photoions and photoelectrons are detected simultaneously then this can allow the disentangling of the electronic population dynamics from the coupled vibrational dynamics [22, 23]

The previously discussed Born - Oppenheimer approximation can break down when energy states get very close to one another and cross. In this so-called non-adiabatic

case, the vibrational (nuclear) and electronic movements are coupled [24], and molecules can cross from one electronic state to another (see Figure 1.9 and Figure 1.4) without absorbing or emitting a photon. As energy must be conserved, when a molecule passes from one electronic state to a lower state this way, the resultant state is highly vibrationally excited. In the liquid or condensed phase, this vibrational energy is often dissipated through the surrounding molecules as heat. In the gas phase, however, this can often lead to "statistical" dissociation of elements of the molecule. These radiationless relaxation routes are a common feature of excited state photochemistry [25-27]. The advent of femtosecond spectroscopy [28] has allowed some of these processes to be studied in a way not possible before [2, 8].

Figure 1.9 demonstrates a conical intersection. As two potential energy surfaces get close to each other, there is often a point at one particular configuration of the molecule, where the states become degenerate and cross. This coupling allows rapid radiationless transitions to occur. The name conical intersection comes from the observation that if the potential energy surfaces are plotted as a function of two nuclear coordinates, they will form inverted cones centred at the degeneracy point (see Figure 1.9).

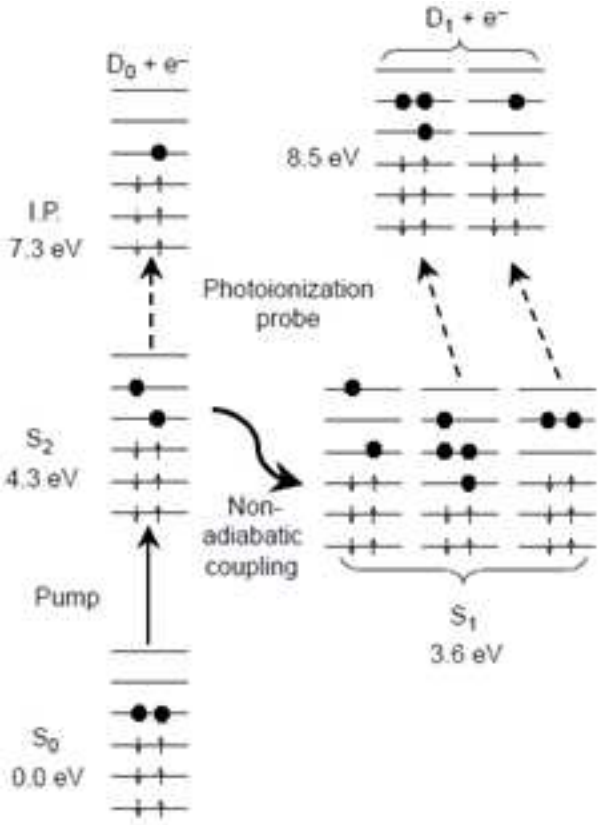


Figure 1.8: Cartoon of molecular orbital configurations and photoionization correlations in a pump-probe experiment. The initial pump photon excites the molecule to the S_2 excited state. The molecule evolves through a conical intersection into another (S_1) excited state. The initially excited state correlates with the D_0 cationic ground state, while the S_1 state correlates with the D_1 cationic excited state. After ionisation, the cation state gives an indication of which previous excited states the molecule was in. From [21].

It has been found that non-adiabatic processes can play a particularly important role within biomolecules, protecting them from the harmful effects of UV radiation. Many biological molecules are hetero-aromatic and contain oxygen or nitrogen atoms in addition to carbon and hydrogen (see, for example, Figure 1.2). Sobolewski and Domcke [15]

proposed a general mechanism for relaxation after photo-excitation for these molecules (Figure 1.4). There is generally an anti-bonding σ* orbital along the N-H or O-H bond. The πσ* state that this creates is generally low lying, often one of the lowest three singlet excited states in the molecule. It generally has a very low oscillator strength, and thus is not usually involved in initial photo-absorption, however the πσ* state does often have a conical intersection with strongly absorbing lower lying states. Once populated, the state is repulsive along the N-H or O-H co-ordinate and has two main methods of decay, through H atom dissociation along repulsive co-ordinate, or through ultrafast internal conversion to the ground state via a further conical intersection at extended bond distances. The σ* orbitals are shown for indole and 5-hydroxyindole in Figure 1.3. The nodes along the N-H and O-H bonds can be clearly seen. These demonstrate the anti-bonding nature of the orbital along these co-ordinates.

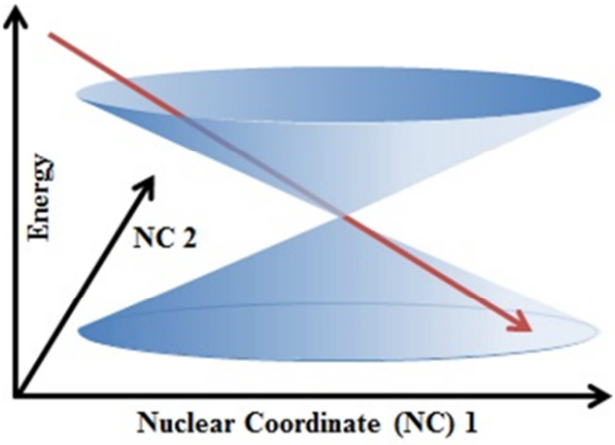


Figure 1.9: Cartoon of a conical intersection. The potential energy surfaces of two electronic states along two molecular co-ordinates has a point of degeneracy. This allows radiationless transfer from one electronic state to another, i.e an electron or hole can move from one orbital to another without needing to absorb or release any energy.

Ashfold and co-workers [25] recently published a review of the experimental and theoretical work conducted on this topic that shows the πσ* state active in many different molecules of biological importance such as phenols and azoles, which are building blocks of larger biological systems such as DNA and amino acids.

1.3. Molecular Dynamics Experimental Techniques

1.3.1. Exploring Dynamics Using Spectroscopy

Various forms of spectroscopy have been used since the 19th century as a method of identifying and understanding the energy levels and structure of atoms and molecules [29]. Spectroscopy may be used to gain information about the molecular orbitals and molecular dynamics through direct observations of the molecules themselves. Performing spectroscopy on an ultrafast timescale allows one to directly observe the ultrafast processes that occur within the molecule. Just as there are many different

methods of doing conventional spectroscopy, each yielding different information about the molecule, there are also many methods of exploring dynamics through spectroscopy.

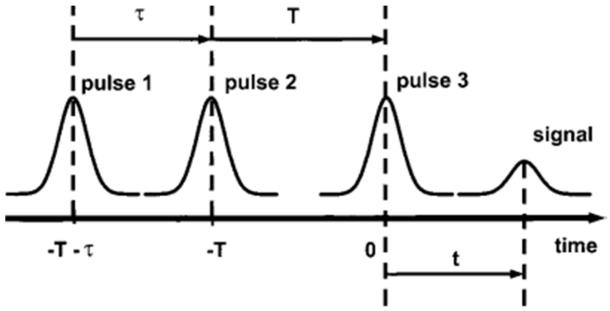


Figure 1.10: The pulse scheme of a 2D spectroscopy experiment. Three pulses with successive delays τ and T are applied to the system. The time origin is conventionally set to the middle of the third pulse. The photon echo signal arises at times t>0. In 2D spectroscopy the first delay τ is varied to record a two-dimensional signal (in τ and t) for a given delay T. Taken from [32].

As laser pulses become shorter, the bandwidth of the pulse becomes larger, as is explained in more detail in Section 1.4. For this reason, as the time resolution of an experiment increases, the frequency resolution decreases. There is therefore an inherent trade-off that must be made when examining ultrafast dynamics (those that occur on a sub-picosecond timescale). Measurements that have the ability to observe the timescales of the processes are unable to provide accurate energies for these processes. For this reason, in order to fully understand the timescales and electronic states involved in ultrafast dynamics, often more than one method must be utilised. Frequency-resolved and time-resolved measurements are complementary in nature; combining the results of different experiments on the same molecule can give a much deeper understanding of dynamical processes that are occurring. Frequency-resolved studies probe the asymptotic distribution and are state resolved. Time-resolved studies give “real time” information, at the price of lower energy resolution. Some of the more common methods for exploring molecular dynamics, both time- and frequency-resolved, are briefly summarised below.

Transient absorption spectroscopy [30] firstly excites the molecules of interest using an ultrafast laser pulse of a known central wavelength, usually chosen to excite to a particular energy level or group of energy levels. A precisely controlled time later an ultrafast white light pulse passes through the sample and the absorption of this pulse is measured on a spectrometer. The white light pulse can be generated by using a fast xenon arc flash lamp [30] or more commonly by focussing an ultrafast pulse into an optical medium, where self-phase modulation makes a white light continuum [31]. By changing the time delay between the two pulses a picture of how the relative population of different energy levels evolves immediately after excitation can be extracted.

Transient two dimensional (2D) Spectroscopy [34, 35] is a variation of this approach that explores the coupling between different electronic states. By systematically varying the time delay between three pulses, and detecting the emissions (see Figure 1.10), a 2D absorption / emission frequency picture is built up. Information about the energies of each state that is both absorbing and emitting photons lies along the diagonal, where absorption frequency equals emission frequency (orange line in Figure 1.11). Information about how these states couple to each other lies in the cross peaks (see Figure 1.11). For example, if the molecule absorbs at one wavelength then evolves to a new state and emits at a different wavelength, this will show as a peak that is not on the diagonal. This allows one to understand how the energy flows from one electronic state to another. This method is implemented using a train of ultrafast pulses used to excite the molecules and stimulate emission. The Fourier transform is used to change the pulse's relative time delays into frequencies. Observing how the 2D spectrum evolves over time after excitation can be done by altering the time delay of the third pulse. This method is mainly used to study vibrational transitions in the ground state, using infrared pulses, however this method can be used with visible and ultraviolet pulses to study excited state transitions [35].

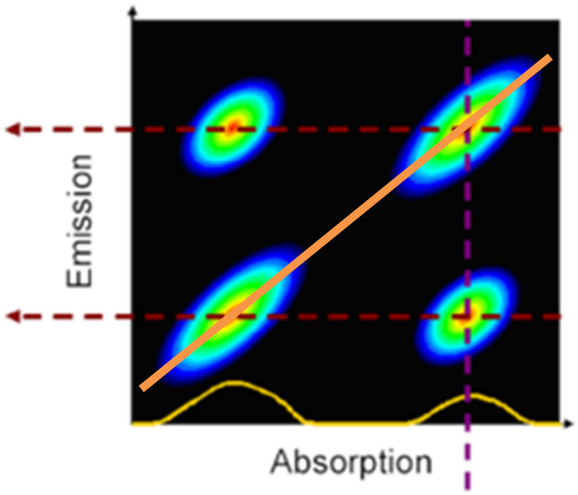


Figure 1.11: 2D Spectroscopy gives a spectrum with two frequency axes. Absorption frequency is along one axis and emission frequency along the other. Taken from reference [33].

Laser Induced Fluorescence [10] photo-excites a molecule and then observes the fluorescence emitted by it. By scanning the excitation wavelength one can observe where the fluorescing excited states lie. This method can be studied on the ultrafast time scale [36] using, for example, time correlated single photon counting methods [37]. This measures the timescale of fluorescence after photo-excitation with picosecond resolution. This method is useful for studying excited states that decay through fluorescence, but cannot observe any states that decay through other methods such as bond breaking or internal conversion.

Ultrafast Electron Diffraction [38] firstly optically excites a molecule to a known electronic state, then uses a time delayed ultrafast electron beam to observe the

structural evolution of the molecule. By comparing the diffraction pattern to theory, the occupied electronic state can be deduced at the different time delays and the molecular dynamics can be observed. As electrons travel through space, they repel each other and so spread out. This creates difficulties in creating ultrafast pulses of electrons. Currently the time resolution is limited to the picosecond regime for this method, but it provides useful complimentary information to optical pump-probe work.

Multi-mass ion imaging [39] is a useful technique to probe the translational energies of photo-fragments. In this technique the molecule is excited with a precisely determined wavelength and then dissociates. Some time later the fragments are ionised and the ions accelerated towards a detector through a cylindrical energy analyser. This results in a 2D image with the translational (kinetic) energy on one axis and the mass on the other axis. This technique gives a time resolution of only microseconds, but the photo-fragment energy resolution can be used to give information on dissociation events happening on a much shorter time scale and the high excitation frequency resolution can be used to identify thresholds and barriers for different dissociation processes [40-44].

H-atom Rydberg Tagging [45, 46] also gives information on the translational energies of photo-fragments. After photo-dissociation, the fragments of interest are excited to a very high-lying Rydberg state. They then travel under field-involevedfree conditions towards a detector, where they are ionised and detected. The time-of-flight yields the translational energy of the fragment. The photo-excitation wavelength and polarisation angle can be altered to give more information on the photo-dissociation mechanisms, such as thresholds for photo-dissociation processes and anisotropy of photo-fragment distributions. Ashfold's group have used this method very successfully on many molecules, including those of relevance to this thesis [47-58].

Resonantly enhanced multi-photon ionisation (REMPI) [59] can be used in a variety of different setups to extract information both in the time domain and the frequency domain. A molecule can be directly ionised with more than one photon; however if there is an electronic state lying at the same energy as the photon energy, then the molecule is much more likely to experience this multi-photon ionisation. This process is called resonance enhancement. In the simplest case one can scan the wavelength of the photolysis laser and record the number of ions, thus gaining a high resolution spectrum of the excited states within the molecule [60]. In addition, by altering the polarisation of the laser, one can probe the dissociation products vector correlations, which can be very sensitive to the evolution of dynamics over a potential energy surface [61]. A popular

special case of REMPI is zero electron kinetic energy (ZEKE) photoelectron spectroscopy [62]. In this case the wavelength is scanned and numbers of molecules excited to high lying Rydberg states are detected. This is done by waiting some nanoseconds for directly ionised electrons to disperse, and subsequently field ionising the Rydberg electrons and detecting them. This gives very high resolution excited state spectra and is useful for precise determinations of ionisation potentials. REMPI can also be used as a probe in a time-resolved setting. By setting the wavelength to a known transition one can selectively ionise (and thus detect) only the molecule of interest in a mixed sample. This can be useful for detecting the fragments of a photo-dissociation event [63, 64]. By scanning the time delay between photo-excitation of the molecule and photo-ionisation of the fragment one can detect the time scale of dissociation after excitation. This method has been used extensively by the Stavros group, including some studies on systems relevant to this thesis [64-69]

1.3.2. Time-Resolved Photoelectron Spectroscopy

Time-resolved photoelectron [6] (TRPES) and photoion [70] spectroscopy methods are yet another special case of the REMPI technique. In this case, the molecule is firstly prepared in an excited electronic state using an ultrafast laser pulse. At some time delay later the molecule is ionised using a second ultrafast pulse and the resultant electron or positive ion is detected (see Figure 1.12). By initially exciting to specific electronic states and scanning the delay time between the two pulses, the lifetime of the states can be extracted, as well as the lifetimes of any subsequent states the molecule passes through during the relaxation process. Ion yield experiments, as practised by Longarte [71] and Radloff's [72] groups, are easier to implement, and can be mass separated, so that signals from different molecules, or different sizes of clusters, can be separated from one another.

Photoelectron measurements, as conducted here and practiced by Neumark [6], Stolow [21], Fielding [73], Ullrich [74], and Suzuki's [75] groups, have the advantage of giving extra information about the kinetic energies of the electrons. This kinetic energy gives us a measure of the electronic state within the molecule before it was ionised. Figure 1.13 shows a typical dataset using this method. This technique is more differential than ion yield techniques as a spectrum of photoelectron energies is collected at each time delay. This can reveal the individual processes occurring and add extra information to assist in interpreting the results. For example, in Figure 1.13 the

initially excited state is decaying into a new electronic state that results in higher energy photoelectrons. If one were to simply look at the total yield of electrons or ions over time, no change would be observed. In addition to this, in imaging experiments the angular direction of photoelectron travel can yield information on the symmetries of the electronic state undergoing ionisation. This gives yet another method of differentiating between different states, as will be expanded upon in Section 3.3.3.

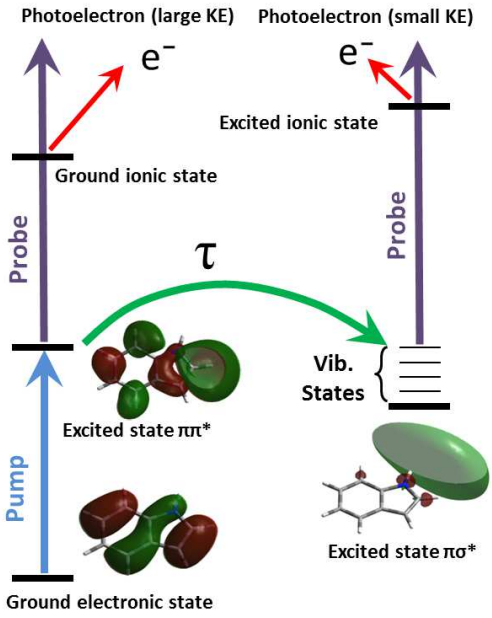


Figure 1.12: Cartoon of energy levels, showing the mechanisms involved in time resolved photoelectron spectroscopy. Firstly the pump excites a molecule to an excited state, which over some time τ may evolve into another excited state. Some time later a probe will ionise the molecule. If it is in the initially excited state the electron will have a certain kinetic energy. If the molecule has moved to a different excited state, due to Koopman's correlations the molecule will project to a different ionised electronic state, resulting in an electron with a different kinetic energy. Measuring the evolution of these energies gives us information about molecular dynamics.

An advantage of this method is that the probe photon will in principle ionise any populated state above a certain level, regardless of symmetry based selection rules. This means that optically dark states that neither absorb nor fluoresce can be directly observed. However, this is limited due to Franck-Condon considerations, as explained in Section 1.2.2, so not every state can be ionised with a given probe energy. Increasing the probe energy can be advantageous in allowing more states to be ionised, as will be discussed later in Chapter 5.

Various methods have been utilized to measure the kinetic energies and/or the angular directions of the photoelectrons or photoions. Time of flight has been, and still is, a very popular method [76]. One detection method uses magnetic bottle analysers – which have a collection efficiency of 50% or better [76-79] to collect the charged particles and pull them to a detector. The magnetic bottle uses an intense magnetic field to parallelise the particles, then a weaker magnetic field to guide them towards the detector. More details on the magnetic bottle can be found in Section 2.2. An advantage of the magnetic bottle is that it is one of the methods that allows for photoelectron -

photoion coincidence measurements to be taken. This is where two separate detectors are run simultaneously, one measuring the photoelectron energy and arrival time, the other the photoion energy, arrival time and mass. If the pump and probe intensities are turned down to such a level that there is less than one ionisation event per laser shot, then by recording the time of arrival of the ion and electron, each photoelectron can be matched to its corresponding cation, hence co-incidence detection. This means the photoelectron signals from different species can be separated. The magnetic bottle gives accurate measures of the spread of kinetic energies, but does not give any information about the angular direction in which the electrons were emitted. Photoelectron or photoion imaging [63] overcomes this drawback in that it provides information about the angular distribution of the electrons in addition to the translational energy information. The next section discusses imaging techniques.

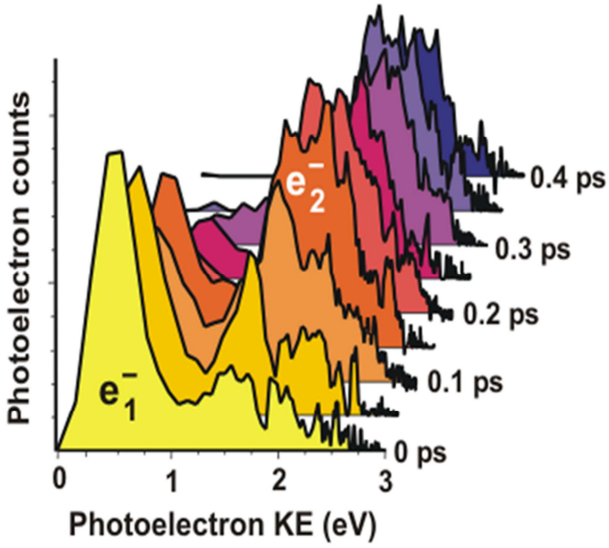


Figure 1.13: Typical TRPES result, showing pump-probe delay time along one horizontal axis, and the energy along the other. The vertical axis shows signal intensity. This demonstrates an initially excited state decaying into another longer lived state. Taken from [21].

1.3.3. Charged Particle Imaging

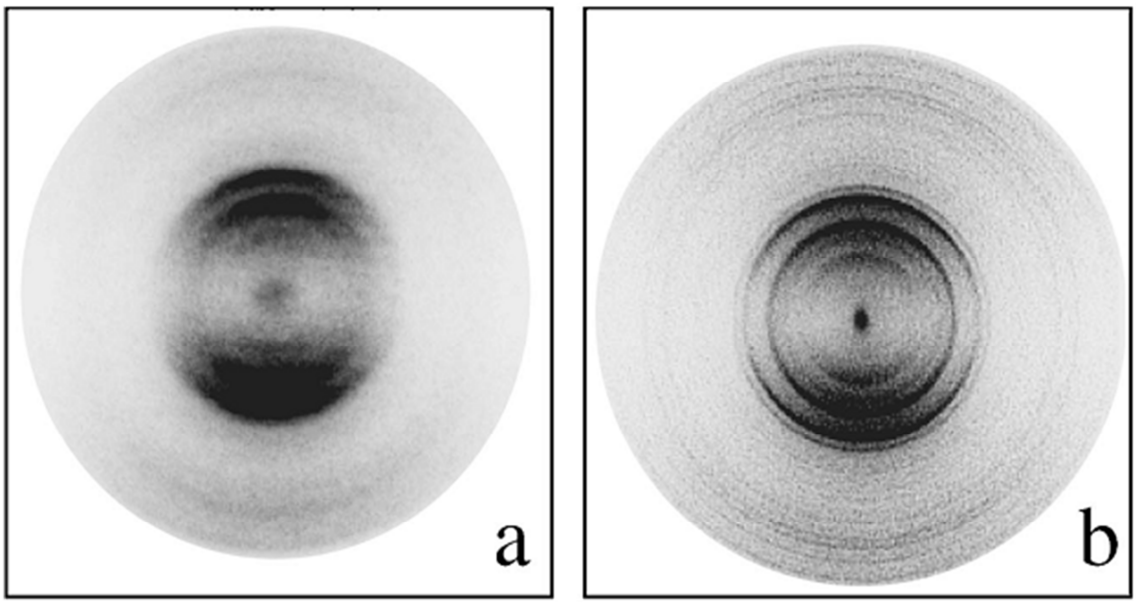


Figure 1.14: (a) O^+ ion image measured with the fine mesh grid; (b) the same with imaging lens. Taken from [59].

Simple imaging techniques were first demonstrated by Solomon and co-workers in the 1960s. They conducted experiments inside a glass bulb coated in phosphor [80], and were able to view the angular distribution of photofragments by observing which sections of the phosphor lit up. In 1986 Chandler and Houston used a new method [63] to investigate two-dimensional (2-D) velocity projections of photodissociation products. This method measures the 2D projection of the 3D charged particle cloud, and detects almost 100% of the emitted charged particles. The angular information is useful in gaining insight into the molecular dynamics. For example in some cases a change in electronic state within the molecule does not change the kinetic energy of the resultant electrons, but does change the direction of travel, due to the symmetries of the molecular orbitals involved. This additional information can assist in de-convoluting and interpreting the various excited state dynamics occurring within a complex molecular system [81-83]. This methods used for this are expanded upon in Chapter 3.

When a group of molecules or atoms are ionised, the photoelectrons create a set of expanding concentric spheres. Each sphere contains charged particles of a certain kinetic energy and corresponds to a peak on a standard photoelectron spectrum. When these concentric spheres hit the detector they are projected into 2D circles. By measuring the diameter of the circle, the kinetic energy of the charged particles can be extracted. In addition, a measure of the angular distribution of the charged particles can be obtained by fitting expected distributions to the data. For example, if an electronic state is more likely to emit fragments along

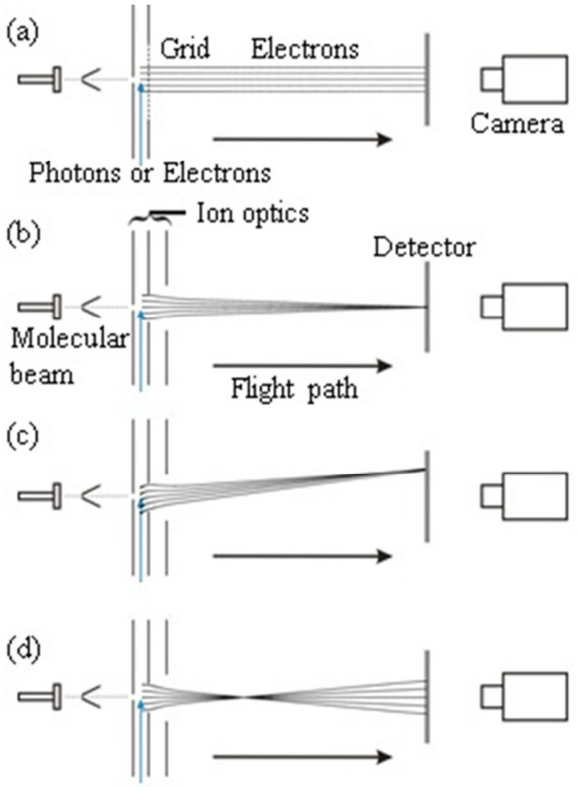


Figure 1.15: Illustration of how velocity map imaging work. (a) grid electrode accelerates electrons on to the detector but has no focussing effect (b) ion optics focus all the electrons with the same initial velocity vector to a single spot (c) another example of (b) but with the electrons initial velocity vector lying perpendicular to the flight axis. (d) ion optics voltage ratio must be chosen carefully to avoid over or under focussing. This is an example of over-focussing.

the axis of symmetry, the sphere will be denser around the axis, and less dense perpendicular to the axis, giving the resultant circle the characteristic distribution seen in Figure 1.14. Processing this distribution to extract spectral information is discussed in the next section.

In order to image these spheres, electrodes are placed to accelerate the charged particles towards a detector, rather like in a cathode ray tube. As the spheres travel towards the detector they continue to expand, so the size of the resultant circles can be altered by changing the accelerating voltages, therefore changing the time of travel. A simple phosphor screen is not a very good detector as a single charged particle does not create a spot that is bright enough to detect. By putting dual microchannel plates before the phosphor screen, a single charged particle will create a chain reaction resulting in a group of several million electrons hitting the phosphor screen at a specific location. A camera can then detect the location of each spot [63]. Another form of detector known as a delay-line anode uses a wire grid instead of a phosphor screen [85, 86]. As the group of electrons hit the grid it causes a current spike in the nearest horizontal and vertical wires. The position and time that the particle arrived can be accurately determined this way by monitoring the current spikes in the wires. This method can also be used for coincidence measurements as described in the previous section, as the time that the particle hits the detector is very precisely known.

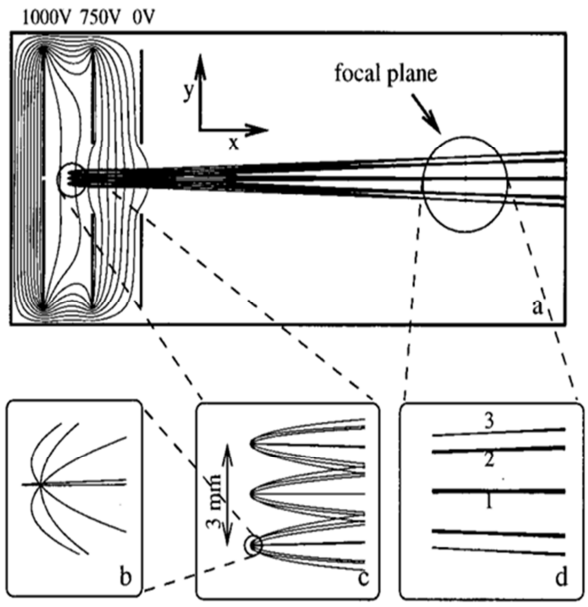


Figure 1.16: From [59] Simulated (Simion 6.0) equipotential surfaces and ion trajectories for a velocity map imaging lens setup. (a) shows the overall view and (b-d) display details. Since the molecular beam has a finite width, three points (c) are chosen to represent the spread in starting position. From each point eight ions are ejected (b) with 45^o angle spacing. By the focussing plane (d) all the ions with the same ejection angle have come together, regardless of origin. 1 corresponds to ejection angles of 180^o and 0^o, 2 to 45° and 3 to 90°.

In 1997 velocity map imaging (VMI) was introduced by Eppink and Parker [84] and has developed into wide use since then [87-92]. In standard imaging, when a molecular beam is ionised, an electric field accelerates the ions or electrons towards a detector, thus imaging the angular and kinetic energy

information. This was originally done using a repelling cathode and an accelerating anode with a wire grid section in it to allow the electrons to pass through (see Figure 1.15(a)). The molecular beam has a finite radius however, and this spreading can cause blurring in the image (see Figure 1.14(a)) as each charged particle is originating from a slightly different location. In VMI the electric field caused by the high voltages placed on the ion optics is curved in such a way that all the ions emitted from a finite volume with the same velocity are focussed onto the same spot on the detector, much like an optical lens will focus light (see Figure 1.16). Changing the voltages on the lenses allow the focussing conditions to be changed and optimised (see Figure 1.15(b-d)). This allows for much greater energy resolution (as the detected position becomes a function of only kinetic energy and angular direction, not dependant on original position (see Figure 1.14(b)).

A variation on velocity map imaging is slice imaging [88-91]. This technique uses a series of lenses to stretch the cloud of charged particles out along the time of flight axis. This creates a time difference between the first and last ions to strike the detector. If the time difference is long enough then the detector can be gated so that it only detects the ions in a particular region of the charge cloud. This can be used to collect the slice through the centre of the distribution, thus removing requirement to process the images as is discussed in the next section. This is an advantage as inverting the images is computationally expensive, can introduce noise, and is prone to human error due to incorrect centre-point assignment. In addition, this removes the requirement discussed in the next section that the cloud must be cylindrically symmetric, allowing different experimental setups, including removing the polarisation restrictions on the pump and probe beams. This technique has been successfully used for ion imaging, however is not currently possible for electron imaging due to the larger velocities involved.

1.3.4. Charged Particle Image Processing

The images obtained using VMI consist of the 2D projection of the 3D particle distribution. In order to obtain the original 3D distribution, the assumption is normally made that the distribution is cylindrically symmetric with the symmetry axis lying parallel to the 2D detector surface. This is a good assumption if the polarisation vectors of the pump and probe photons are parallel both to each other and to the 2D detector surface.

With this knowledge we can then begin to look at ways of extracting the 3D distribution from the 2D projection techniques (see Figure 1.17 for an example). An overview of many of the (older) methods of reconstructing the 3D distribution can be found in a review by Whittaker and co-workers [93] and references therein. The cylindrically symmetric 3D charged particle distribution can be denoted by $I(\rho, z) \equiv I(x, y, z)$ where ρ is the radius and z is the altitude along the axis of symmetry. The projection of this distribution onto the detector (y,z) plane can be written as:

$$P(y, z) = \int I(\rho, z)\, dx \tag{1.4}$$

To obtain the original $I(\rho, z)$ from the measured $P(y, z)$ the inverse Abel transform may be used:

$$I(\rho, z) = -\frac{1}{4\pi} \int_r^R \frac{P(y, z)}{\sqrt{y^2 - \rho^2}}\, dy \tag{1.5}$$

This equation is not easy to solve. One problem with Equation 1.5 is that as ρ^2 approaches y^2 the denominator approaches zero. This causes amplification of any noise in the image and this can lead to large distortions in the reconstructed image, often giving a stripe of noise along the centre of the image. There are several methods used to either solve this equation, or otherwise deduce the 3D distribution from the 2D projection.

Backtracking or “onion peeling” techniques [94-97] start with the most energetic charged particle (i.e. right at the outer edge of the image), and calculate what contribution particles with this energy would make to a 2D projected image. This is then subtracted from the image, and the resultant image is iteratively re-processed using the same technique. This effectively “peels” the velocity distribution from the original image, ring by ring.

Basis set expansion (BASEX) [98-100] makes Equation 1.2 easier to solve by expanding $P(y, z)$ over a set of easily analysed basis functions. The expansion coefficients used then yield all the information required to obtain the original $I(\rho, z)$. This method can avoid the centreline noise distortions by first converting to polar coordinates (pBASEX). In this configuration the noise is distributed to the edges of the

reconstructed image, where it causes fewer problems in subsequent analysis. This method is commonly used by the scientific community, but can be quite slow, especially to analyse multiple images.

Pattern recognition techniques, developed by Loock and others [101], fits the 2D distribution with equations such as those described in Chapter 3 (Eqs 3.28 and 3.30) to extract the velocity distribution and anisotropy directly from the 2D distribution. This method effectively avoids distortions due to noise amplification and eliminates the need for further data processing on the inverted image.

Iterative forward convolution methods, developed by Vrakking and others [100, 102], start with an initial guess of the 3D distribution $I(\rho,z)$, and a new 2D image $P(y,z)$ is calculated using Equation 1.4 and compared to the initial image. A correction is applied and a new 3D distribution generated; the process is repeated until the differences between the calculated and original 2D images are acceptably small. One way of doing this [102] takes advantage of the similarity between the 3D distribution and the 2D image by using the 2D image as the initial guess. This iterative process avoids problems of centre-line noise, as instead the noise is projected towards the centre spot. The Fourier moment analysis technique developed by Brouard and Vallance [103] uses a similar method but has the advantage that it does not require the 3D distribution to be cylindrically symmetric. This technique expresses the 2D projection as a Fourier series in angular coordinates and uses forward convolution to fit basis image functions to the 2D projection. It extracts bipolar moments that characterise the vector correlations between the transition dipole moment of the parent molecule and the velocity and angular distributions of the photoelectrons or photoions.

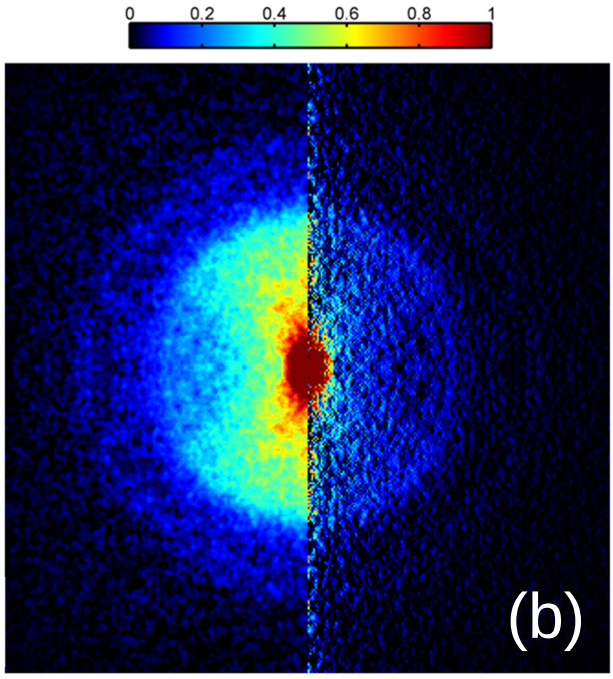


Figure 1.17: Example of reconstruction using the matrix Abel inversion method (a) 2D projection P(y,z) from phenol data set and (b) Slice through centre of reconstructed 3D distribution I(ρ,z). From [81].

One further method of rapidly solving Equation 1.5 uses matrix inversion techniques (see Figure 1.17) [104, 105], which have already been the subject of much optimisation work. More detail on this technique can be found in Section 3.3.2: Initial Data Processing. This method gives around two orders of magnitude speed improvement over the other methods detailed here, and can process an entire data set comprising 44 images, each 400 × 400 pixels, in around a second on a standard personal computer (2.67 GHz processer).

The many methods for reconstructing the 3D distribution each have their own advantages and disadvantages. Size of image, signal to noise ratio, symmetry, energy resolution, data to be extracted from the 3D distribution, and number of images to be processed all have a bearing in choosing the most appropriate method for a given application. Once the 3D distribution has been extracted, generally the radial distribution $I(\rho)$ and the anisotropy parameters β_n are required for a full analysis of the molecular information. These can be found using the following equations [105]:

$$I(\rho) = \int I(\rho, \theta)\, \rho \sin(\theta)\, d\theta \tag{1.6}$$

$$I(\rho, \theta) = \frac{\beta_0(\rho)}{4\pi} \left(1 + \sum_n \beta_n(\rho)\, P_n(\theta)\right) \tag{1.7}$$

The valid values for n are dependent upon the light used to excite and ionise the molecules; in the case of two photons with linear polarisation, n = 2,4. This is the case for all the experiments in this thesis, and the equation can be simplified to:

$$I(\rho, \theta) = \frac{\beta_0(\rho)}{4\pi} \left(1 + \beta_2(\rho)\, P_2(\theta) + \beta_4(\rho)\, P_4(\theta)\right) \tag{1.8}$$

where P_n is the n^{th} order Legendre polynomial and θ is the angle from the axis of symmetry. The intensity as a function of energy can be calculated from the intensity as a function of radius (see Section 3.3.3 for details) to yield a photoelectron or photoion spectrum. With the energy and anisotropy of each ring identified, the rings can be assigned and the molecular dynamics investigated.

1.3.5. Studying Molecules in the Gas Phase

In general we wish to study the molecules of interest in the gas phase. Although in Nature biological molecules are generally found in the condensed phase, it is much more demanding to interpret results in this phase as charged particles cannot be cleanly extracted, and there are other molecules contaminating the results. In the gas phase we can study the dynamics of the isolated molecules. We can also apply much more differential techniques in this phase, allowing us to de-convolute different processes, and gain a much deeper understanding into the photochemistry of the molecular systems. It can be a challenge to get model biological systems into the gas phase as many are large molecules, solid and not volatile at room temperature. They need to be inserted into the interaction region of the vacuum chamber in such a way as to reliably get a pure source of individual molecules (not clusters) and without consuming too many of the sometimes expensive molecules. There are various methods of doing this.

The molecular beam [106-108] is a common choice. The sample in its gaseous form expands into the vacuum through a narrow nozzle. Often an inert carrier gas can be utilised to carry the molecules into the gas phase. The inert gas is passed through the solid or liquid sample, which is often heated to raise the vapour pressure. This results in a gaseous mixture of sample and carrier. For photo-electron work the inert gas should have a high ionisation threshold to avoid confusing the experimental results. The expansion into the chamber cools the particles down to extremely low temperatures in the initial high pressure portion of the beam, through collisions with the carrier gas. Once past this they then travel in a beam with very few collisions, as demonstrated in Figure 1.18

The Even Lavie valve [109] is a high pressure pulsed valve that can produce very short molecular beam pulses. The short pulses are useful in that only a small amount of sample is used for each pulse, reducing running costs when dealing with hard to synthesise molecules and reducing pumping requirements.

The helium nanodroplets technique is a variation on the molecular beam.

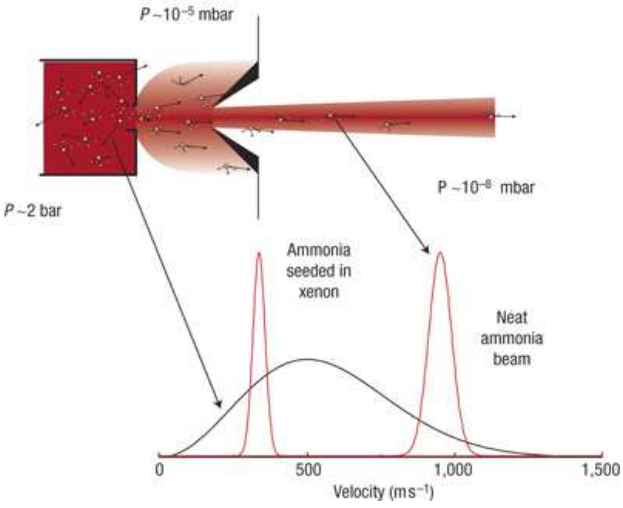


Figure 1.18: Diagram showing a molecular beam expanding through a skimmer, and the velocities of the molecules within different parts of the beam. Taken from [108].

Helium is expanded through a nozzle to create droplets. These droplets then pass through a pick-up region containing a low density vapour of the molecule of interest. The molecules dope the droplets which, as the droplets evaporate, cool the molecules [110, 111] to extremely low temperatures.

Electrospray ionisation [112, 113] is a method that can be used for molecules too large to easily enter the gas phase. A solution of the sample is pushed through a small capillary to a conical highly charged nozzle creating a fine charged aerosol. The solvent evaporates, transferring the charge to the molecule. The result is highly charged ions in the gas phase. This is a valuable method of getting molecules into the gas phase, but is of limited use to those wishing to study neutral species.

Laser desorption ionisation [114, 115] involves ionising a molecule using a laser pulse. Often this is assisted with a matrix that absorbs the radiation, and transfers the energy to the molecule. This avoids damaging the molecule. The matrix crystal and molecule are stationary on a target and the ions are drawn into the gas phase using an electric field. This method cannot be used for photoelectron spectroscopy, as the molecules are highly charged.

Matrix assisted laser desorption/ionisation (MALDI) [116, 117] is a technique to get fragile molecules into the gas phase. The molecules of interest are dissolved into a matrix of crystallised matrix molecules (often aromatic acids). The matrix is irradiated by a UV laser pulse, which ablates a portion of the matrix. This generally leads to a mixture of ionised and neutral molecules of interest, and matrix molecules. This can be useful for techniques that have the ablity to distinguish between different molecular masses, but not for photoelectron spectroscopy where the results could become contaminated by signal from photoionisation of the matrix molecules.

Laser induced acoustic desorption (LIAD) [114, 118-120] is a much cleaner and gentler method of desorbing large molecules. Here the molecule of interest is deposited in a metal foil. The back of the foil is irradiated by a laser pulse, which stimulates an acoustic wave. This wave propagates through the foil and results in the sample desorbing from the surface in a low density plume. This method is expanded upon in Section 5.2.2 as it is promising for our future studies.

Once the molecules are in the gas phase, the ultrafast laser pulses must be correctly timed and positioned to interact with the molecules, and the detector system must also be timed correctly to collect the signal. The correct combinations of these methods give many powerful tools for exploring the dynamics of molecules.

1.4. Femtosecond Lasers

1.4.1. Creating Ultrafast Pulses

Since the first demonstration of the laser 50 years ago [121], they have often been used as tools to explore the chemical world. The advent of ultrafast lasers [122] has led to many new techniques that explore the chemical processes that occur on short time scales [8], as has previously been discussed in this chapter. The duration of the pulse of light used in time-resolved photoelectron spectroscopy is the limiting factor on the time resolution of the experiment [73]. Changing the time delay between the "pump" and the "probe" pulses allows the time dependent decay processes to be studied. However, as the pulses have finite durations, there will be a small spread in the time delay experienced by each molecule. In order to reduce this spread, a shorter pulse is preferable. The Heisenberg uncertainty principle states that the energy and position of a wave cannot be known to an arbitrary precision simultaneously. This is expressed as $\Delta E\ \Delta t \geq \frac{\hbar}{2}$. As the position of the wave is known very precisely in the ultrafast limit, the energy (or wavelength) must therefore not be precisely known. This means that a short pulse must be composed of a many wavelengths, or in other words have a large bandwidth. The bandwidth of the pulse is inversely proportional to the duration of the pulse, with the shortest pulses being effectively white light [123]. The minimum pulse duration of a laser is generally determined by the bandwidth of the gain medium. The

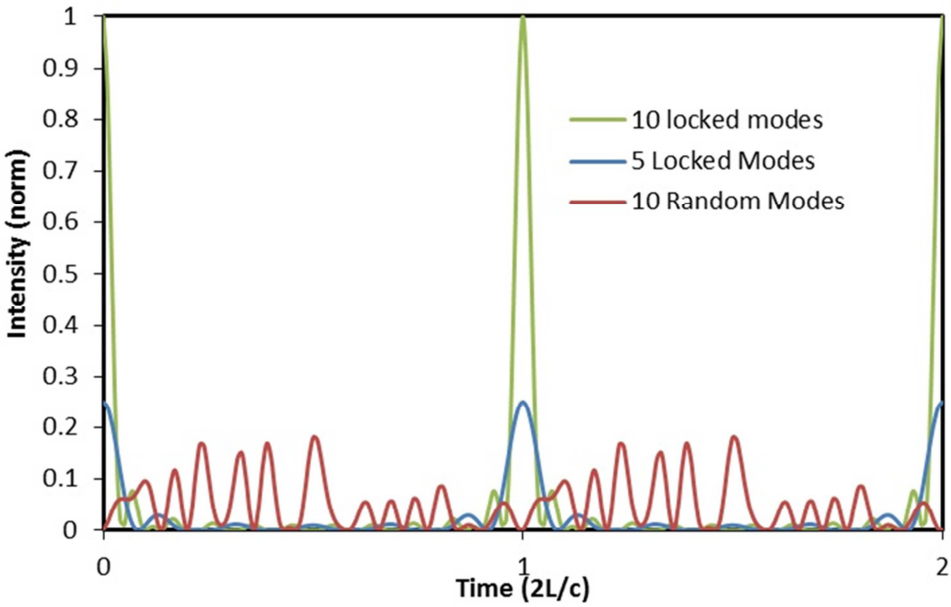


Figure 1.19: An example of the effect of modelocking on laser temporal profile using a small number of modes. The random profile represents the laser in continuous wave mode, while the two modelocked profiles demonstrates the change effected by fixing the phase. Note that as the number of modes increases from 5 to 10, the pulse intensity increases and the pulse duration decreases.

time-bandwidth product is given by

$$\Delta v \Delta t \geq K \tag{1.9}$$

where Δt is the pulse duration, $\Delta \nu$ is the frequency bandwidth, and K is a constant that depends on the pulse shape (0.441 for a Gaussian profile, which is typical for our experiment). For a transform-limited pulse (minimum duration for a given bandwidth) the temporal profile is the Fourier transform of the spectral profile.

Inside a laser, only certain wavelengths λ can be amplified, depending on the path length L inside the cavity:

$$L = \frac{m\,\lambda}{2}, \quad m = 1,2,3,\ldots \tag{1.10}$$

where m is an integer. Wavelengths that satisfy Equation 1.6 are referred to as modes. In order to have a broad bandwidth, many modes must be simultaneously amplified within the cavity, with ultrafast pulses typically containing ~10^4 modes [4]. If these lasing modes are summed randomly then the power distribution will also be random. If, however, the phase between the modes is "locked" (see Figure 1.19), then the peak power becomes much larger and the random fluctuations between the peaks reduce towards zero. The resultant pulses become shorter as more modes are included, allowing ultrafast pulses to be achieved when thousands of modes are locked. A laser may be passively modelocked due to the optical Kerr effect [124]. This effect occurs when intense light passes through a material; the electric field causes an electric polarisation in the material, and so changes the refractive index, causing self-focussing within it. As the phase randomly changes between the different modes, power fluctuations will occur. When these power fluctuations become large enough, they will cause self-focussing to occur. A slit can be employed to select only the focussed light (see Figure 1.20), thus allowing amplification of the modes that are in phase and attenuating all other modes. This passive effect can be used to reliably maintain modelocking within a cavity.

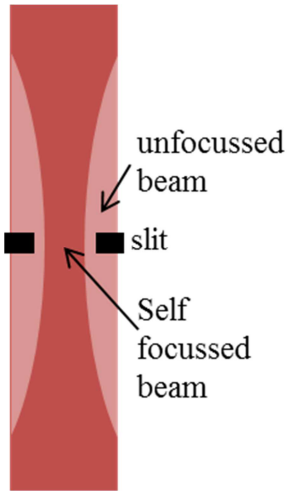


Figure 1.20: Cartoon of passive modelocking due to the optical Kerr effect. See text for details.

Often small fluctuations in cavity length must be made to create the initial power fluctuations that lead to modelocking. That can be done by vibrating the gain medium or tapping the end of the cavity. As the pulse travels through optics inside the cavity, the different modes can become separated, as the refractive index of optical materials, and so the group velocity of the light inside the material, varies as a function of wavelength. When the frequencies are spread in this way the pulse is said to be "chirped" (see Figure 1.21). Prisms within the cavity can be used to compensate by creating the correct amount of negative chirp for each round trip (see Figure.1.22 and Figure 1.23).

Until the early 1990s, the shortest pulses available came from dye lasers. These lasers have broad bandwidth gain mediums and can be used to produce pulses typically down to 30 fs. Dye lasers can be problematic due to dye degrading with time, thermal effects, and flow problems. The most common ultrafast lasers in use today are solid state titanium doped sapphire (Ti:Sapphire, Figure 1.23), due to the large gain bandwidth (approx 600 nm-1100 nm) and the ability to be pumped with frequency-doubled Nd:YAG lasers (532 nm). This laser system is able to produce pulses shorter than 5 fs [125, 126], pulse energies larger than 10 mJ, and repetition rates of several kHz.

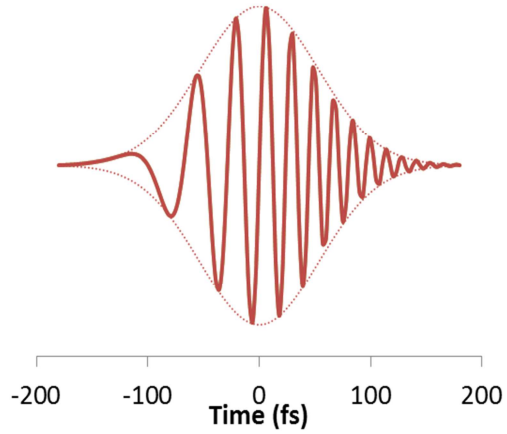


Figure 1.21: Cartoon of a chirped pulse.

Ultrafast pulses by their very nature contain all their energy in a very short time, and so tend to have high peak power. In order to amplify such a pulse, which is crucial for the highly non-linear frequency conversion processes required for ultraviolet pump-probe photoelectron experiments, care must be taken to protect any optics and crystals from damage. Chirped pulse amplification, developed in the 1980s [127-135], overcomes this problem by stretching the pulse duration before the amplification process, thus bringing the peak power below the damage

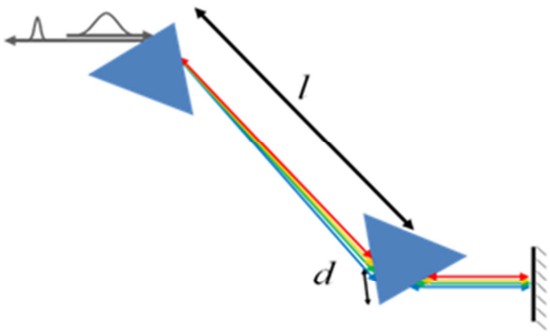


Figure 1.22: *Schematic of a Prism Compressor.*

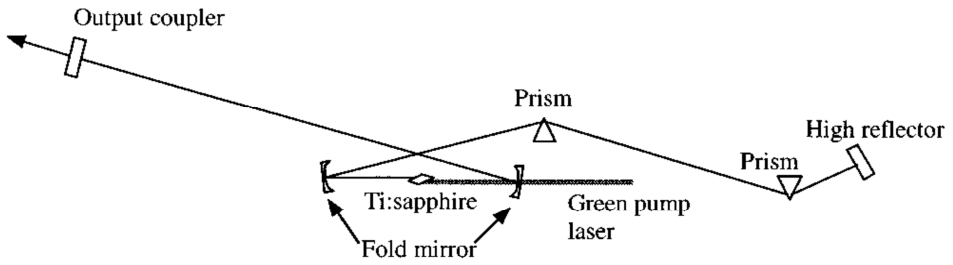


Figure 1.23: Schematic diagram of a self-modelocked Ti:sapphire laser [123].

threshold of the sensitive optics. Once the pulse has been amplified, it is recompressed to be close to its original length. The pulse is stretched and compressed by chirping: dispersing the pulse so that shorter wavelengths are delayed more than longer wavelengths or vice versa. The stretcher and compressor make use of negative dispersion techniques such as diffraction gratings or prism pairs (see Figure 1.22). Care must be taken in choosing the gain medium, as if the bandwidth of the pulse is narrowed, then the pulse will not compress back to its original duration. There is often a trade-off between power and pulse duration due to this effect. In most cases the same type of gain medium is used as that originally used in the laser oscillator. The chirped pulse amplifier allows extremely high peak power pulses to be generated, often in the terawatt regime [127].

1.4.2. Non-Linear Optics

Non-linear optics were made possible by the advent of lasers in 1960. Laser light's extreme intensities allowed the development of this new range of optical phenomena, as was first demonstrated by Weinreich and co-workers in 1961 [136]. Non-linear optics gives the ability to change the wavelength of a laser beam, and create tuneable wavelength ultrafast pulses.

Non-linear optics are crucial in providing a range of wavelengths from an ultrafast laser system. The amplified Ti:Sapphire laser is tuneable only over a very small range (typically around 760-840 nm), however, to probe the excited state dynamics of many molecules we need pulses in the UV range (200-400 nm). As light passes through a medium, a dielectric polarisation $\vec{P}$ is set up [137]:

$$\vec{P} = \varepsilon_0\left(\chi^{(1)}\,\vec{E} + \chi^{(2)}\,\vec{E}^2 + \chi^{(3)}\,\vec{E}^3 + \cdots\right) \tag{1.11}$$

were $\chi^{(i)}$ are the i^{th} order susceptibilities, ε_0 is the permittivity of free space and $\vec{E}$ is the laser electric field vector. In most everyday interactions the electric field intensity is so

low that only the first order susceptibility $\chi^{(1)}$ is important, accounting for such optical phenomena as refraction and reflection. Within laser beams, however, the intensity can be so high that second or higher order effects can become important. These effects are collectively known as non-linear optics.

The electric field vector of a beam of light made up of two frequencies ($\omega_1$and ω_2) with amplitudes ($A_1$and A_2) can be expressed as:

$$\vec{E} = A_1 e^{-i\omega_1 t} + A_2 e^{-i\omega_2 t} + cc \tag{1.12}$$

Substituting 1.7 into 1.6 we can arrive at the following equation for the second order dielectric polarisation:

$$\vec{P}^{(2)} = \chi^{(2)}\,\vec{E}^2 = \chi^{(2)}\Big(2(|A_1|+|A_2|) + A_1^2 e^{-i\mathbf{2\omega_1}t} + A_2^2 e^{-i\mathbf{2\omega_2}t} + 2A_1A_2 e^{-i(\boldsymbol{\omega_1}+\boldsymbol{\omega_2})t} + 2A_1A_2^* e^{-i(\boldsymbol{\omega_1}-\boldsymbol{\omega_2})t}\Big) \tag{1.13}$$

This equation demonstrates the second order effects that can occur in a non-linear medium. The first (blue) term expresses simple optical reflection. The second and third (red) terms demonstrate second harmonic generation (doubling): if the incident light has a frequency ω_i, light with twice that frequency ($2\omega_i$) is emitted. The fourth (green) term demonstrates sum-frequency generation (mixing): the light that is emitted has a frequency equal to the sum of both the incident frequencies ($\omega_1 + \omega_2$). The final (orange) term demonstrates difference-frequency generation: the emitted light has a frequency equal to the difference in the two incident frequencies ($\omega_1 - \omega_2$). These processes allow for many different wavelengths to be generated from a single frequency source.

In order for these processes to occur, the different wavelengths must be phase matched inside the non-linear medium. This means ensuring a proper phase relationship between the interacting waves through the course of propagation through the medium. This will only be the case when the refractive index is the same for all the wavelengths involved in the process, i.e. $n(\omega_1) \equiv n(\omega_2)$. This is impossible in most optical media, and if the condition is not met, then the generated pulse will get destroyed due to destructive interference before it exits the crystal (see Figure 1.24). The birefringence of non-linear crystals can be exploited to create phase matching conditions for different

wavelength combinations. The refractive index of a birefringent crystal is dependent on the temperature of the crystal and the polarisation of the wave. In addition, the angle of the crystal affects the refractive index for the extraordinary (s-polarised) beam, but not the ordinary (p-polarised) beam. For most cases, an angle can be found that allows the different beams to stay in phase with each other. There are different types of phase matching, in the case of type I, the initial frequencies must have the same polarisation, which is perpendicular to the output frequency. This is the case in all of the doubling and mixing setups described in the next sections. There is also type II phase matching, where the initial frequencies have perpendicular polarisations. This often results in lower output powers than type I [137].

As has been described in the previous section, the group velocity inside a medium is a function of the wavelength. Some wavelengths will travel much slower than others inside the non-linear crystal, causing "walk-off", or temporal separation of the different pulses, which can cause power loss and beam profile distortions. The effects of walk-off can be minimized if the crystal is kept to a thickness such that the walk-off is less than the pulse duration; however, the efficiency of the frequency mixing process is directly proportional to the square of the crystal thickness. For these reasons the thickness of the crystal used must be chosen with care. Thinner crystals can create shorter pulses, but often with lower power, while thicker crystals can sometimes create higher power pulses, but only if the wavelengths involved are not too disparate from each other. Some examples of using non-linear optics within our lab in Heriot-Watt University can be found in Section 3.1.

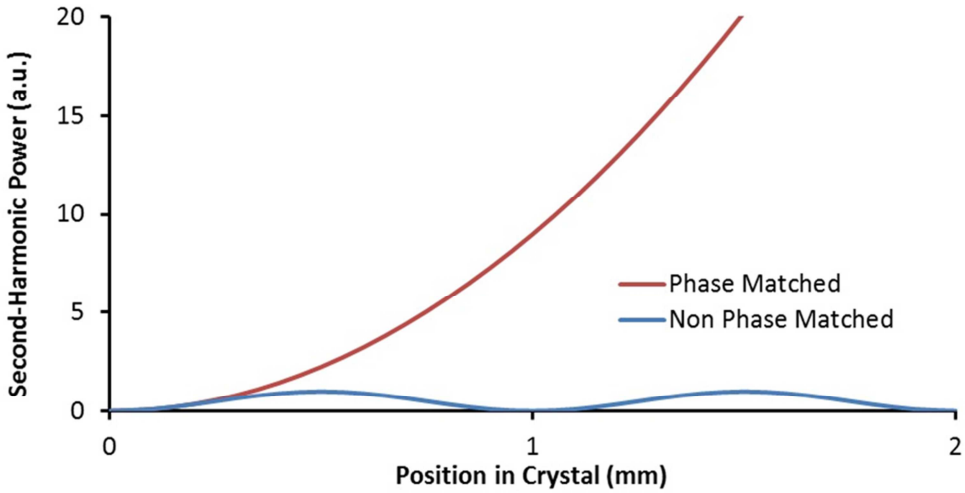


Figure 1.24: Graph illustrating the effect of phase matching. The curves show the growth of second harmonic power along the propagation direction inside the crystal. The red curve illustrates the phase matched case, where the power grows in proportion to the propagation distance squared, and the blue line illustrates the non phase matched case, where the power fluctuates between zero and a small value.

1.4.3. Characterising Ultrafast Pulses

Once a femtosecond pulse is produced, the duration or the phase of the pulse must then be measured. Not even the most rapid photodetector is able to react in the timescales involved. In theory, the pulse needs to be measured using a device that has a faster response than the event to be measured. As yet, there are no controllable events faster than the pulse duration that could be used as a measure, other than the pulse itself, or another pulse produced by a similar laser. There are various methods employed that overlap two or more pulses of light, in order to measure the length of one of them [125, 138-144].

The autocorrelation [141] technique crosses two identical pulses spatially and temporally in a non-linear material (see Figure 1.25). The number of second harmonic photons that are generated is proportional to the square of the intensity, so when the pulses are temporally overlapped many more photons are generated. The second harmonic light produced can be measured as a function of the time delay between the two pulses. This gives a measure of the length of the pulse but not the phase.

Frequency-resolved optical grating (FROG) [138-140] is a spectrally resolved autocorrelation technique. It involves measuring the spectrum and intensity of a signal pulse as a function of the delay between the two pulses, using nonlinear materials. The resultant trace is a frequency-time plot like a musical score (Figure 1.26). Grating-eliminated no-nonsense observation of ultrafast incident laser light e-fields (GRENOUILLE) [145] is a variation on the FROG. The alignment sensitive beam splitter and delay stage is replaced by a lens and Fresnel biprism that splits the pulse into two, and focuses each resultant pulse to a horizontal stripe on the crystal. The two pulses are automatically overlapped on each other, and the delay between the two is a function of distance along the stripe. This one-shot technique is much less sensitive to alignment problems. The crystal used is much thicker than that used in FROG. The different elements of the beam hitting the crystal are coming from different input angles due to the tight focus. The limited, angle dependant,

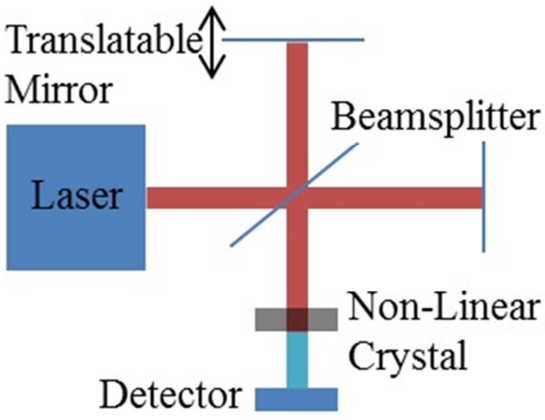


Figure 1.25: Schematic of a simple, Michelson Interferometer based, Autocorrelation setup.

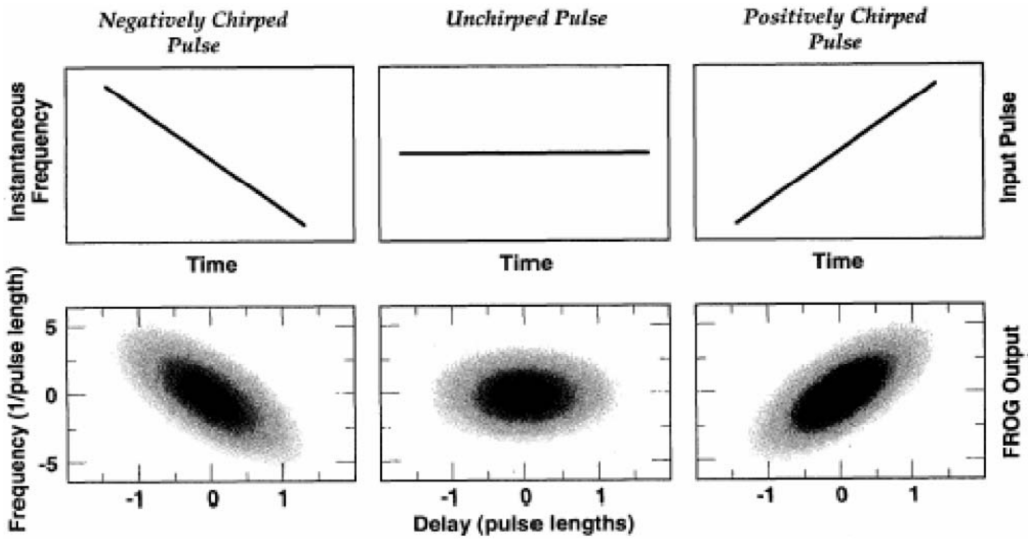


Figure 1.26: FROG traces for negatively chirped, un-chirped, and positively chirped Gaussian pulses. The top row shows the frequency versus the delay between the two laser pulses. The figures in the bottom row show the FROG traces as density plots (black indicates high intensity and white indicates low intensity). From [73].

phase matching bandwidth of the nonlinear crystal causes the wavelength that is generated to vary with incident angle. This replaces the spectrometer in the FROG and the resultant beam, after focussing, has the time along the horizontal axis, and the wavelength along the vertical axis, much like a FROG trace. All that is required to extract a trace is for a camera to observe the final beam profile.

Spectral interferometry (SI) [146, 147] uses a reference pulse that has already been characterised. If the two pulses are overlapped in space, with a small time delay between them, then they will interfere to create a spectrum that has an oscillating profile. The phase of the pulse can then be extracted using a Fourier transform. This method requires no non-linear optics so is useful for low intensity pulses.

Spectral phase interferometry for direct electric-field reconstruction (SPIDER) [142-144] uses SI interferometry techniques to find the phase without requiring a well characterised light pulse at the start. The pulse is split in three parts. One part is chirped to create a much longer pulse. Of the other two, one is slightly delayed with respect to the other. These three pulses are then combined in a non-linear crystal to create two new, frequency doubled, pulses. The delayed pulse will be combined with a bluer section of the chirped pulse to create a slightly bluer doubled pulse. The interference pattern between the two, spectrally shifted, resultant pulses, reveals both the temporal intensity and phase. This technique requires no moving parts and can be done on a single-shot basis. It is employed widely for use with infrared and visible pulses, and can be modified for use with blue and UV pulses by using difference frequency mixing rather than frequency doubling.

1.5. Summary and Overview of Thesis

Spectroscopy is a powerful tool for understanding the electronic states within atoms and molecules. When this is combined with time resolution, the ability to observe how these states evolve after a certain initial event gives us much deeper insight into how chemistry happens. Without the work of thousands coming before us, we would never be able to achieve the current level of understanding. This thesis explores only one small aspect of this much larger picture.

This thesis starts by looking at my time-resolved photoelectron spectroscopy work conducted in Ottawa in Albert Stolow's lab at the National Research Council Canada. This following chapter looks at indole and 5-hydroxyindole in an effort towards understanding 5,6-dihydroxyindole, which is a component of the melanin polymer. The experimental setup is described briefly and a full discussion of the results is laid out. Chapter 3 discusses the setup of the new time-resolved photoelectron spectroscopy laboratory in Heriot-Watt University, Scotland. This chapter goes into the details of the experimental equipment used; the software that was developed to control the experiment and analyse the data; the optical setup; and the spectrometer. Chapter 4 describes the first results obtained using this equipment. Phenol and the phenol derivatives catechol, resorcinol and hydroquinone were studied in order to shed light on the photo-excitation dynamics of these molecules, and to understand the effect on the dynamics that the hydroxyl substituent can have on a nearby hydroxyl group. Finally, in Chapter 5, possible future directions for the lab in Heriot-Watt University are explored and the different sections of the thesis are tied together.

1.6. References

[1] A. H. Zewail, *Femtochemistry: Atomic-Scale Dynamics of the Chemical Bond*, J. Phys. Chem. A, **104**, 5660, (2000).

[2] I. V. Hertel and W. Radloff, *Ultrafast Dynamics in Isolated Molecules and Molecular Clusters*, Rep. Prog. Phys., **69**, 1897, (2006).

[3] H. R. Hudock, B. G. Levine, A. L. Thompson, H. Satzger, D. Townsend, N. Gador, S. Ullrich, A. Stolow, and T. J. Martinez, *Ab Initio Molecular Dynamics and Time-Resolved Photoelectron Spectroscopy of Electronically Excited Uracil and Thymine*, J. Phys. Chem. A, **111**, 8500, (2007).

[4] S. Link, H. A. Durr, and W. Eberhardt, *Femtosecond Spectroscopy*, J. Phys. - Condens. Mat., **13**, 7873, (2001).

[5] D. Townsend and K. L. Reid, *Photoionization Dynamics Probed by Angle-Resolved Photoelectron Spectroscopy of $NH_3(\widetilde{B}^1E'')$*, J. Chem. Phys., **112**, 9783, (2000).

[6] A. Stolow, A. E. Bragg, and D. M. Neumark, *Femtosecond Time-Resolved Photoelectron Spectroscopy*, Chem. Rev., **104**, 1719, (2004).

[7] A. H. Zewail, *Laser Femtochemistry*, Science, **242**, 1645, (1988).

[8] L. R. Khundkar and A. H. Zewail, *Ultrafast Molecular Reaction Dynamics in Real-Time: Progress Over a Decade*, Ann. Rev. Phys. Chem., **41**, 15, (1990).

[9] J. M. Hollas, *Vapour-Phase Ultra-Violet Absorption Spectra of Indene, Indole, Coumarone and Thionaphthene*, Spectrochim. Acta., **19**, 753, (1963).

[10] J. L. Kinsey, *Laser-Induced Fluorescence*, Ann. Rev. Phys. Chem., **28**, 349, (1977).

[11] L. SerranoAndres and B. O. Roos, *Theoretical Study of the Absorption and Emission Spectra of Indole in the Gas Phase and in a Solvent*, J. Am. Chem. Soc., **118**, 185, (1996).

[12] H. Satzger, D. Townsend, M. Z. Zgierski, S. Patchkovskii, S. Ullrich, and A. Stolow, *Primary Processes Underlying the Photostability of Isolated DNA Bases: Adenine*, P. Natl. Acad. Sci. USA, **103**, 10196, (2006).

[13] P. Meredith and T. Sarna, *The Physical and Chemical Properties of Eumelanin*, Pigm. Cell. Res., **19**, 572, (2006).

[14] N. Glasser and H. Lami, *Nonradiative Decay of Indoles under Collision-Free Conditions*, J. Chem. Phys., **74**, 6526, (1981).

[15] A. L. Sobolewski, W. Domcke, C. Dedonder-Lardeux, and C. Jouvet, *Excited-State Hydrogen Detachment and Hydrogen Transfer Driven by Repulsive $^1\pi\sigma^*$ States: A New Paradigm for Nonradiative Decay in Aromatic Biomolecules*, Phys. Chem. Chem. Phys., **4**, 1093, (2002).

[16] S. N. Dixit and V. Mckoy, *Theory of Resonantly Enhanced Multiphoton Processes in Molecules*, J. Chem. Phys., **82**, 3546, (1985).

[17] R. Livingstone, O. Schalk, A. E. Boguslavskiy, G. R. Wu, L. T. Bergendahl, A. Stolow, M. J. Paterson, and D. Townsend, *Following the Excited State Relaxation Dynamics of Indole and 5-Hydroxyindole Using Time-Resolved Photoelectron Spectroscopy*, J. Chem. Phys., **135**, 194307, (2011).

[18] M. Born and J. R. Oppenheimer, *Zur Quantentheorie Kontinuierlicher Spektren*, Ann. Phy., **41**, 457, (1927).

[19] J. Franck and E. G. Dymond, *Elementary Processes of Photochemical Reactions*, Trans. Farad. Soc., **21**, 536, (1926).

[20] P. W. Atkins and R. S. Friedman, *Molecular Quantum Mechanics*, Oxford University Press, Oxford, (2005).

[21] V. Blanchet, M. Z. Zgierski, T. Seideman, and A. Stolow, *Discerning Vibronic Molecular Dynamics Using Time-Resolved Photoelectron Spectroscopy*, Nature, **401**, 52, (1999).

[22] V. Blanchet, M. Z. Zgierski, and A. Stolow, *Electronic Continua in Time-Resolved Photoelectron Spectroscopy. I. Complementary Ionization Correlations*, J. Chem. Phys., **114**, 1194, (2001).

[23] M. Schmitt, S. Lochbrunner, J. P. Shaffer, J. J. Larsen, M. Z. Zgierski, and A. Stolow, *Electronic Continua in Time-Resolved Photoelectron Spectroscopy. Ii. Corresponding Ionization Correlations*, J. Chem. Phys., **114**, 1206, (2001).

[24] R. Renner, *Zur Theorie der Wechselwirkung zwischen Elektronen- und Kernbewegung bei dreiatomigen, stabförmigen Molekülen*, Z.Phys. A Had. Nuc., **92**, 172, (1934).

[25] M. N. R. Ashfold, G. A. King, D. Murdock, M. G. D. Nix, T. A. A. Oliver, and A. G. Sage, *$\pi\sigma^*$ Excited States in Molecular Photochemistry*, Phys. Chem. Chem. Phys., **12**, 1218, (2010).

[26] C. Z. Bisgaard, H. Satzger, S. Ullrich, and A. Stolow, *Excited-State Dynamics of Isolated DNA Bases: A Case Study of Adenine*, ChemPhysChem, **10**, 101, (2009).

[27] G. A. Worth and L. S. Cederbaum, *Beyond Born-Oppenheimer: Molecular Dynamics through a Conical Intersection*, Annu. Rev. Phys. Chem., **55**, 127, (2004).

[28] A. H. Zewail, *Femtochemistry: Atomic-Scale Dynamics of the Chemical bond using ultrafast lasers - (Nobel lecture)*, Angew. Chem. Int. Edit., **39**, 2587, (2000).

[29] G. Kirchhoff and R. Bunsen, *Chemische Analyse durch Spectralbeobachtungen*, Ann. Phy., **186**, 161, (1860).

[30] R. Bonneau, J. Wirz, and A. D. Zuberbuehler, *Methods for the Analysis of Transient Absorbance Data*, Pure. Appl. Chem., **69**, 979, (1997).

[31] D. A. V. Kliner, J. C. Alfano, and P. F. Barbara, *Photodissociation and Vibrational Relaxation of I_2- in Ethanol*, J. Chem. Phys., **98**, 5375, (1993).

[32] A. V. Pisliakov, T. Mancal, and G. R. Fleming, *Two-Dimensional Optical Three-Pulse Photon Echo Spectroscopy. II. Signatures of Coherent Electronic Motion and Exciton Population Transfer in Dimer Two-Dimensional Spectra*, J. Chem. Phys., **124**, 234505, (2006).

[33] T. Brixner, *www.phys-chemie.uni-wuerzburg.de/en/arbeitsgruppen/lehrstuhl_i_prof_t_brixner/home/research/2d_spectroscopy/*, (2012).

[34] I. Noda, *Generalized Two-Dimensional Correlation Method Applicable to Infrared, Raman, and Other Types of Spectroscopy*, Appl. Spectrosc., **47**, 1329, (1993).

[35] T. Brixner, J. Stenger, H. M. Vaswani, M. Cho, R. E. Blankenship, and G. R. Fleming, *Two-Dimensional Spectroscopy of Electronic Couplings in Photosynthesis*, Nature, **434**, 625, (2005).

[36] J. Kasparian, M. Rodriguez, G. Méjean, J. Yu, E. Salmon, H. Wille, R. Bourayou, S. Frey, Y.-B. André, A. Mysyrowicz, R. Sauerbrey, J.-P. Wolf, and L. Wöste, *White-Light Filaments for Atmospheric Analysis*, Science, **301**, 61, (2003).

[37] B. Gompf, R. Günther, G. Nick, R. Pecha, and W. Eisenmenger, *Resolving Sonoluminescence Pulse Width with Time-Correlated Single Photon Counting*, Phys. Rev. Lett., **79**, 1405, (1997).

[38] J. C. Williamson, J. Cao, H. Ihee, H. Frey, and A. H. Zewail, *Clocking Transient Chemical Changes by Ultrafast Electron Diffraction*, Nature, **386**, 159, (1997)

[39] S.-T. Tsai, C.-K. Lin, Y. T. Lee, and C.-K. Ni, *Multimass Ion Imaging Detection: Application to Photodissociation*, Rev. Sci. Instrum., **72**, 1963, (2001).

[40] M. F. Lin, C. M. Tseng, Y. T. Lee, and C. K. Ni, *Photodissociation Dynamics of Indole in a Molecular Beam*, J. Chem. Phys., **123**, 124303 (2005).

[41] C. M. Tseng, Y. M. Choi, C. L. Huang, C. K. Ni, Y. T. Lee, and M. C. Lin, *Photodissociation of Nitrosobenzene and Decomposition of Phenyl Radical*, J. Phys. Chem. A, **108**, 7928, (2004).

[42] C. M. Tseng, Y. T. Lee, and C. K. Ni, *H Atom Elimination from the πσ* State in the Photodissociation of Phenol*, J. Chem. Phys., **121**, 2459, (2004).

[43] C. M. Tseng, Y. T. Lee, M. F. Lin, C. K. Ni, S. Y. Liu, Y. P. Lee, Z. F. Xu, and M. C. Lin, *Photodissociation Dynamics of Phenol*, J. Phys. Chem. A, **111**, 9463, (2007).

[44] C. M. Tseng, Y. T. Lee, C. K. Ni, and J. L. Chang, *Photodissociation Dynamics of the Chromophores of the Amino Acid Tyrosine: p-Methylphenol, p-Ethylphenol, and p-(2-Aminoethyl)phenol*, J. Phys. Chem. A, **111**, 6674, (2007).

[45] L. Schnieder, W. Meier, K. H. Welge, M. N. R. Ashfold, and C. M. Western, *Photodissociation Dynamics of H2S at 121.6 nm and a Determination of the Potential Energy Function of SH($A^2\Sigma^+$)*, J. Chem. Phys., **92**, 7027, (1990).

[46] X. M. Yang, *State-to-state dynamics of elementary chemical reactions using Rydberg H-atom translational spectroscopy*, Int. Rev. Phys. Chem., **24**, 37, (2005).

[47] M. G. D. Nix, A. L. Devine, B. Cronin, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy of the near UV Photolysis of Indole: Dissociation Via the $^1\pi\sigma^*$ State*, Phys. Chem. Chem. Phys., **8**, 2610, (2006).

[48] M. G. D. Nix, A. L. Devine, B. Cronin, and M. N. R. Ashfold, *Ultraviolet Photolysis of Adenine: Dissociation Via the $^1\pi\sigma^*$ State*, J. Chem. Phys., **126**,124312, (2007).

[49] T. A. A. Oliver, G. A. King, and M. N. R. Ashfold, *Position Matters: Competing O-H and N-H Photodissociation Pathways in Hydroxy- and Methoxy-Substituted Indoles*, Phys. Chem. Chem. Phys., **13**, 14646, (2011).

[50] M. G. D. Nix, A. L. Devine, B. Cronin, and M. N. R. Ashfold, *Ultraviolet Photolysis of Adenine: Dissociation via the $^1\pi\sigma^*$ State*, J. Chem. Phys., **126**, 124312, (2007).

[51] G. A. King, T. A. A. Oliver, M. G. D. Nix, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy Studies of the Ultraviolet Photolysis of Phenol-d(5)*, J. Phys. Chem. A, **113**, 7984, (2009).

[52] R. N. Dixon, T. A. A. Oliver, and M. N. R. Ashfold, *Tunnelling Under a Conical Intersection: Application to the Product Vibrational State Distributions in the UV Photodissociation of Phenols*, J. Chem. Phys., **134**, 194303, (2011).

[53] M. G. D. Nix, A. L. Devine, R. N. Dixon, and M. N. R. Ashfold, *Observation of Geometric Phase Effect Induced Photodissociation Dynamics in Phenol*, Chem. Phys. Lett., **463**, 305, (2008).

[54] M. N. R. Ashfold, A. L. Devine, R. N. Dixon, G. A. King, M. G. D. Nix, and T. A. A. Oliver, *Exploring Nuclear Motion through Conical Intersections in the UV Photodissociation of Phenols and Thiophenol*, P. Natl. Acad. Sci. USA, **105**, 12701, (2008).

[55] A. L. Devine, M. G. D. Nix, B. Cronin, and M. N. R. Ashfold, *Near-UV Photolysis of Substituted Phenols, I: 4-fluoro-, 4-chloro- and 4-bromophenol*, Phys. Chem. Chem. Phys., **9**, 3749, (2007).

[56] G. A. King, A. L. Devine, M. G. D. Nix, D. E. Kelly, and M. N. R. Ashfold, *Near-UV Photolysis of Substituted Phenols*, Phys. Chem. Chem. Phys., **10**, 6417, (2008).

[57] M. G. D. Nix, A. L. Devine, B. Cronin, R. N. Dixon, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy Studies of the near Ultraviolet Photolysis of Phenol*, J. Chem. Phys., **125**, 133318, (2006).

[58] G. A. King, T. A. A. Oliver, R. N. Dixon, and M. N. R. Ashfold, *Vibrational Energy Redistribution in Catechol During Ultraviolet Photolysis*, Phys. Chem. Chem. Phys., **14**, 3338, (2012).

[59] J. C. Miller, R. Compton, M. G. Payne, and W. Garrett, *Resonantly Enhanced Multiphoton Ionization and Third-Harmonic Generation in Xenon Gas*, Phys. Rev. Lett., **45**, 114, (1980).

[60] E. J. Bieske and O. Dopfer, *High-Resolution Spectroscopy of Cluster Ions*, Chem. Rev., **100**, 3963, (2000).

[61] M. L. Costen, R. Livingstone, K. G. McKendrick, G. Paterson, M. Brouard, H. Chadwick, Y. P. Chang, C. J. Eyles, F. J. Aoiz, and J. Klos, *Elastic Depolarization of OH(A) by He and Ar: A Comparative Study*, J. Phys. Chem. A, **113**, 15156, (2009).

[62] K. Muller-Dethlefs and E. W. Schlag, *High-Resolution Zero Kinetic Energy (ZEKE) Photoelectron Spectroscopy of Molecular Systems*, Ann. Rev. Phys. Chem., **42**, 109, (1991).

[63] D. W. Chandler and P. L. Houston, *Two-Dimensional Imaging of State-Selected Photodissociation Products Detected by Multiphoton Ionization*, J. Chem. Phys., **87**, 1445, (1987).

[64] G. M. Roberts, A. S. Chatterley, J. D. Young, and V. G. Stavros, *Direct Observation of Hydrogen Tunneling Dynamics in Photoexcited Phenol*, J. Phys. Chem. Lett., **3**, 348, (2012).

[65] A. Iqbal and V. G. Stavros, *Exploring the Time Scales of H-Atom Elimination from Photoexcited Indole*, J. Phys. Chem. A, **114**, 68, (2010).

[66] R. A. L. Smith, V. G. Stavros, J. R. R. Verlet, H. H. Fielding, D. Townsend, and T. P. Softley, *The Role of Phase in Molecular Rydberg Wave Packet Dynamics*, J. Chem. Phys., **119**, 3085, (2003).

[67] K. L. Wells, G. M. Roberts, and V. G. Stavros, *Dynamics of H-Loss in Adenine Via the $^1\pi\sigma^*$ State Using a Combination of ns and fs Laser Spectroscopy*, Chem. Phys. Lett., **446**, 20, (2007).

[68] A. Iqbal, M. S. Y. Cheung, M. G. D. Nix, and V. G. Stavros, *Exploring the Time-Scales of H-Atom Detachment from Photoexcited Phenol-h(6) and Phenol-d(5): Statistical vs Nonstatistical Decay*, J. Phys. Chem. A, **113**, 8157, (2009).

[69] A. Iqbal, L. J. Pegg, and V. G. Stavros, *Direct Versus Indirect H Atom Elimination from Photoexcited Phenol Molecules*, J. Phys. Chem. A, **112**, 9531, (2008).

[70] T. Baumert, M. Grosser, R. Thalweiser, and G. Gerber, *Femtosecond Time-Resolved Molecular Multiphoton Ionization: The Na_2 System*, Phys. Rev. Lett., **67**, 3753, (1991).

[71] B. C. Dian, A. Longarte, and T. S. Zwier, *Hydride Stretch Infrared Spectra in the Excited Electronic States of Indole and its Derivatives: Direct Evidence for the $^1\pi\sigma^*$ State*, J. Chem. Phys., **118**, 2696, (2003).

[72] T. Freudenberg, W. Radloff, H. H. Ritze, V. Stert, K. Weyers, F. Noack, and I. V. Hertel, *Ultrafast Fragmentation and Ionisation Dynamics of Ammonia Clusters*, Z. Phys. D: At. Mol. Clusters, **36**, 349, (1996).

[73] R. E. Carley, E. Heesel, and H. H. Fielding, *Femtosecond Lasers in Gas Phase Chemistry*, Chem. Soc. Rev., **34**, 949, (2005).

[74] R. Crespo-Otero, M. Barbatti, H. Yu, N. L. Evans, and S. Ullrich, *Ultrafast Dynamics of UV-Excited Imidazole*, ChemPhysChem, **12**, 3365, (2011).

[75] T. Suzuki, *Femtosecond Time-Resolved Photoelectron Imaging*, Ann. Rev. Phys. Chem., **57**, 555, (2006).

[76] J. Steadman and J. A. Syage, *A Paraboloidal Electrostatic Reflector for Molecular-Beam Time-of-Flight Photoelectron Spectrometers*, Rev. Sci. Instrum., **64**, 3094, (1993).

[77] O. Cheshnovsky, S. H. Yang, C. L. Pettiette, M. J. Craycraft, and R. E. Smalley, *Magnetic Time-of-Flight Photoelectron Spectrometer for Mass-Selected Negative Cluster Ions*, Rev. Sci. Instrum., **58**, 2131, (1987).

[78] P. Kruit and F. H. Read, *Magnetic-Field Parallelizer for 2-π Electronspectrometer and Electron-Image Magnifier*, J. Phys. E. Sci. Instrum., **16**, 313, (1983).

[79] I. Powis, T. Baer, C.-Y. Ng, I. Powis, T. Baer, and C.-Y. Ng, eds., *High Resolution Laser Photoionization and Photoelectron Studies*, Wiley, New York, (1995).

[80] J. Solomon, *Photodissociation as Studied by Photolysis Mapping*, J. Chem. Phys., **47**, 889, (1967).

[81] R. A. Livingstone, J. O. F. Thomson, M. Iljina, R. J. Donaldson, B. J. Sussman, and D. Townsend, *Time-Resolved Photoelectron Imaging of Excited State Relaxation Dynamics in Phenol, Catechol, Resorcinol and Hydroquinone*, J. Chem. Phys., **Submitted**, (2012).

[82] T. Horio, T. Fuji, Y.-I. Suzuki, and T. Suzuki, *Probing Ultrafast Internal Conversion through Conical Intersection via Time-Energy Map of Photoelectron Angular Anisotropy*, J. Am. Chem. Soc., **131**, 10392, (2009).

[83] M. Tsubouchi, C. A. de Lange, and T. Suzuki, *Ultrafast Dissociation Processes in the NO Dimer Studied with Time-Resolved Photoelectron Imaging*, J. Elec. Spec., **142**, 193, (2005).

[84] A. T. J. B. Eppink and D. H. Parker, *Velocity Map Imaging of Ions and Electrons Using Electrostatic Lenses: Application in Photoelectron and Photofragment Ion Imaging of Molecular Oxygen*, Rev. Sci. Instrum., **68**, 3477, (1997).

[85] A. Gabriel, *Position Sensitive X-Ray Detector*, Rev. Sci. Instrum., **48**, 1303, (1977). [86]R. Moshammer, M. Unverzagt, W. Schmitt, J. Ullrich, and H.

Schmidt-Böcking, *A 4π Recoil-Ion Electron Momentum Analyzer: A High-Resolution "Microscope" for the Investigation of the Dynamics of Atomic, Molecular and Nuclear Reactions*, Nucl. Instrum. Meth. B, **108**, 425, (1996).

[87] M. N. R. Ashfold, N. H. Nahler, A. J. Orr-Ewing, O. P. J. Vieuxmaire, R. L. Toomes, T. N. Kitsopoulos, I. A. Garcia, D. A. Chestakov, S.-M. Wu, and D. H. Parker, *Imaging the Dynamics of Gas Phase Reactions*, Phys. Chem. Chem. Phys., **8**, 26, (2006).

[88] J. J. Lin, J. G. Zhou, W. C. Shiu, and K. P. Liu, *Application of Time-Sliced Ion Velocity Imaging to Crossed Molecular Beam Experiments*, Rev. Sci. Instrum., **74**, 2495, (2003).

[89] D. Townsend, S. K. Lee, and A. G. Suits, *DC Slice Imaging of CH3Cl Photolysis at 193.3 nm*, J. Phys. Chem. A, **108**, 8106, (2004).

[90] D. Townsend, S. K. Lee, and A. G. Suits, *Orbital Polarization from DC Slice Imaging: S(1D2) Alignment in the Photodissociation of Ethylene Sulfide*, Chem. Phys., **301**, 197, (2004).

[91] D. Townsend, M. P. Minitti, and A. G. Suits, *Direct Current Slice Imaging*, Rev. Sci. Instrum., **74**, 2530, (2003).

[92] C. Vallance, *'Molecular Photography': Velocity-Map Imaging of Chemical Events*, Philos. T. Roy. Soc. A., **362**, 2591, (2004).

[93] A. T. J. B. Eppink, S.-M. Wu, and B. J. Whitaker, in *Imaging in Molecular Dynamics: Technology and Applications* (B. J. Whitaker, ed.), Cambridge University Press, Cambridge, p. 65-112, (2003).

[94] C. Bordas, F. Paulig, H. Helm, and D. L. Huestis, *Photoelectron Imaging Spectrometry: Principle and Inversion Method*, Rev. Sci. Instrum., **67**, 2257, (1996).

[95] G. M. Roberts, J. L. Nixon, J. Lecointre, E. Wrede, and J. R. R. Verlet, *Toward Real-Time Charged-Particle Image Reconstruction Using Polar Onion-Peeling*, Rev. Sci. Instrum., **80**, 053104, (2009).

[96] K. Zhao, T. Colvin, W. T. Hill, and G. Zhang, *Deconvolving Two-Dimensional Images of Three-Dimensional Momentum Trajectories*, Rev. Sci. Instrum., **73**, 3044, (2002).

[97] S. Manzhos and H. P. Loock, *Photofragment Image Analysis Using the Onion-Peeling Algorithm*, Comput. Phys. Commun., **154**, 76, (2003).

[98] V. Dribinski, A. Ossadtchi, V. A. Mandelshtam, and H. Reisler, *Reconstruction of Abel-Transformable Images: The Gaussian Basis-Set Expansion Abel Transform Method*, Rev. Sci. Instrum., **73**, 2634, (2002).

[99] G. A. Garcia, L. Nahon, and I. Powis, *Two-Dimensional Charged Particle Image Inversion Using a Polar Basis Function Expansion*, Rev. Sci. Instrum., **75**, 4989, (2004).

[100] F. Renth, J. Riedel, and F. Temps, *Inversion of Velocity Map Ion Images Using Iterative Regularization and Cross Validation*, Rev. Sci. Instrum., **77**, 033103, (2006).

[101] S. Manzhos and H. P. Loock, *Photofragment Image Analysis Via Pattern Recognition*, Rev. Sci. Instrum., **75**, 2435, (2004).

[102] M. J. J. Vrakking, *An Iterative Procedure for the Inversion of Two-Dimensional Ion/Photoelectron Imaging Experiments*, Rev. Sci. Instrum., **72**, 4084, (2001).

[103] M. J. Bass, M. Brouard, A. P. Clark, and C. Vallance, *Fourier Moment Analysis of Velocity-Map Ion Images*, J. Chem. Phys., **117**, 8723, (2002).

[104] Y. T. Cho and S. J. Na, *Application of Abel Inversion in Real-Time Calculations for Circularly and Elliptically Symmetric Radiation Sources*, Meas. Sci. Technol., **16**, 878, (2005).

[105] B. J. Sussman, *Quantum Control Using the Nonresonant Dynamic Stark Effect*, Doctoral Thesis, **Queens University**, Kingston, (2007).

[106] A. Kantrowitz and J. Grey, *A High Intensity Source for the Molecular Beam. Part I. Theoretical*, Rev. Sci. Instrum., **22**, 333, (1951).

[107] G. B. Kistiakowsky and W. P. Slichter, *A High Intensity Source for the Molecular Beam. Part II. Experimental*, Rev. Sci. Instrum., **22**, 328, (1951).

[108] S. Y. T. van de Meerakker, H. L. Bethlem, and G. Meijer, *Taming Molecular Beams*, Nat. Phys., **4**, 595, (2008).

[109] U. Even, J. Jortner, D. Noy, N. Lavie, and C. Cossart-Magos, *Cooling of Large Molecules Below 1 K and He Clusters Formation*, J. Chem. Phys., **112**, 8068, (2000).

[110] K. K. Lehmann and G. Scoles, *Superfluid Helium - the Ultimate Spectroscopic Matrix?*, Science, **279**, 2065, (1998).

[111] J. P. Toennies and A. F. Vilesov, *Superfluid Helium Droplets: A Uniquely Cold Nanomatrix for Molecules and Molecular Complexes*, Angew. Chem. Int. Edit., **43**, 2622, (2004).

[112] J. B. Fenn, M. Mann, C. K. Meng, S. F. Wong, and C. M. Whitehouse, *Electrospray Ionization for Mass-Spectrometry of Large Biomolecules*, Science, **246**, 64, (1989).

[113] D. D. Ebeling, M. S. Westphall, M. Scalf, and L. M. Smith, *Corona Discharge in Charge Reduction Electrospray Mass Spectrometry*, Anal. Chem., **72**, 5158, (2000).

[114] V. V. Golovlev, S. L. Allman, W. R. Garrett, N. I. Taranenko, and C. H. Chen, *Laser-Induced Acoustic Desorption*, Int. J. Mass. Spec., **169**, 69, (1997).

[115] J. Somuramasami and H. I. Kenttamaa, *Evaluation of a Novel Approach for Peptide Sequencing: Laser-Induced Acoustic Desorption Combined with $P(OCH_3)_2^+$ Chemical Ionization and Collision-Activated Dissociation in a Fourier Transform Ion Cyclotron Resonance Mass Spectrometer*, J. Am. Soc. Mass. Spectr., **18**, 525, (2007).

[116] R. E. Russo, X. Mao, H. Liu, J. Gonzalez, and S. S. Mao, *Laser Ablation in Analytical Chemistry - a Review*, Talanta, **57**, 425, (2002).

[117] M. N. R. Ashfold, F. Claeyssens, G. M. Fuge, and S. J. Henley, *Pulsed Laser Ablation and Deposition of Thin Films*, Chem. Soc. Rev., **33**, 23, (2004).

[118] J. Pérez, L. E. Ramírez-Arizmendi, C. J. Petzold, L. P. Guler, E. D. Nelson, and H. I. Kenttämaa, *Laser-Induced Acoustic Desorption/Chemical Ionization in Fourier-Transform Ion Cyclotron Resonance Mass Spectrometry*, Int. J. Mass. Spec., **198**, 173, (2000).

[119] R. C. Shea, C. J. Petzold, J. Liu, and H. I. Kenttämaa, *Experimental Investigations of the Internal Energy of Molecules Evaporated via Laser-Induced Acoustic Desorption into a Fourier Transform Ion Cyclotron Resonance Mass Spectrometer*, Anal. Chem., **79**, 1825, (2007).

[120] I. Bald, I. Dąbkowska, and E. Illenberger, *Probing Biomolecules by Laser–Induced Acoustic Desorption: Electrons at Near Zero Electron Volts Trigger Sugar–Phosphate Cleavage*, Angew. Chem. Int. Edit., **47**, 8518, (2008).

[121] T. H. Maiman, *Stimulated Optical Radiation in Ruby*, Nature, **187**, 493, (1960).

[122] R. Fork, B. Greene, and C. Shank, *Generation of Optical Pulses Shorter Than 0.1 psec by Colliding Pulse Mode Locking*, Appl. Phys. Lett., **38**, 671, (1981).

[123] S. Backus, C. G. Durfee, M. M. Murnane, and H. C. Kapteyn, *High Power Ultrafast Lasers*, Rev. Sci. Instrum., **69**, 1207, (1998).

[124] E. Masse, *J. Kerr. - Measurements and Law in Electro-Optics*, Phil. Mag., **5e série, t. IX**, 157, (1880).

[125] U. Morgner, F. X. Kartner, S. H. Cho, Y. Chen, H. A. Haus, J. G. Fujimoto, E. P. Ippen, V. Scheuer, G. Angelow, and T. Tschudi, *Sub-Two-Cycle Pulses from a Kerr-Lens Mode-Locked Ti:Sapphire Laser*, Opt. Lett., **24**, 411, (1999).

[126] D. H. Sutter, G. Steinmeyer, L. Gallmann, N. Matuschek, F. Morier-Genoud, U. Keller, V. Scheuer, G. Angelow, and T. Tschudi, *Semiconductor Saturable-Absorber Mirror-Assisted Kerr-Lens Mode-Locked Ti:Sapphire Laser Producing Pulses in the Two-Cycle Regime*, Opt. Lett., **24**, 631, (1999).

[127] P. Maine, D. Strickland, P. Bado, M. Pessot, and G. Mourou, *Generation of Ultrahigh Peak Power Pulses by Chirped Pulse Amplification*, IEEE. J. Quantum. Elect., **24**, 398, (1988).

[128] D. Strickland and G. Mourou, *Compression of Amplified Chirped Optical Pulses*, Opt. Commun., **56**, 219, (1985).

[129] M. Pessot, P. Maine, and G. Mourou, *1000 Times Expansion Compression of Optical Pulses for Chirped Pulse Amplification*, Opt. Commun., **62**, 419, (1987).

[130] M. Pessot, J. Squier, P. Bado, G. Mourou, and D. J. Harter, *Chirped Pulse Amplification of 300-fs Pulses in an Alexandrite Regenerative Amplifier*, IEEE. J. Quantum. Elect., **25**, 61, (1989).

[131] E. B. Treacy, *Optical Pulse Compression with Diffraction Gratings*, IEEE. J. Quantum. Elect., **5**, 454, (1969).

[132] O. E. Martinez, J. P. Gordon, and R. L. Fork, *Negative Group-Velocity Dispersion Using Refraction*, J. Opt. Soc. Am. A, **1**, 1003, (1984).

[133] O. E. Martinez, *Grating and Prism Compressors in the Case of Finite Beam Size*, J. Opt. Soc. Am. B, **3**, 929, (1986).

[134] O. E. Martinez, *Design of High-Power Ultrashort Pulse-Amplifiers by Expansion and Recompression*, IEEE. J. Quantum. Elect., **23**, 1385, (1987).

[135] O. E. Martinez, *3000 Times Grating Compressor with Positive Group-Velocity Dispersion - Application to Fiber Compensation in 1.3-1.6 μm Region*, IEEE. J. Quantum. Elect., **23**, 59, (1987).

[136] P. A. Franken, A. E. Hill, C. W. Peters, and G. Weinreich, *Generation of Optical Harmonics*, Phys. Rev. Lett., **7**, 118, (1961).

[137] R. W. Boyd, *Nonlinear Optics*, Academic Press, New York, (2008).

[138] R. Trabino, *Frequency-Resolved Optical Gating: The Measurement of Ultrashort Laser Pulses*, Kluwer Academic Publishers, Boston, (2000).

[139] R. Trebino, K. W. DeLong, D. N. Fittinghoff, J. N. Sweetser, M. A. Krumbugel, B. A. Richman, and D. J. Kane, *Measuring Ultrashort Laser Pulses in the Time-Frequency Domain Using Frequency-Resolved Optical Gating*, Rev. Sci. Instrum., **68**, 3277, (1997).

[140] D. J. Kane and R. Trebino, *Characterization of Arbitrary Femtosecond Pulses Using Frequency-Resolved Optical Gating*, IEEE. J. Quantum. Elect., **29**, 571, (1993).

[141] A. Brun, P. Georges, G. Lesaux, and F. Salin, *Single-Shot Characterization of Ultrashort Light-Pulses*, J. Phys. D. - Appl. Phys., **24**, 1225, (1991).

[142] C. Dorrer, E. M. Kosik, and I. A. Walmsley, *Direct Space-Time Characterization of the Electric Fields of Ultrashort Optical Pulses*, Opt. Lett., **27**, 548, (2002).

[143] C. Iaconis and I. A. Walmsley, *Spectral Phase Interferometry for Direct Electric-Field Reconstruction of Ultrashort Optical Pulses*, Opt. Lett., **23**, 792, (1998).

[144] C. Iaconis and I. A. Walmsley, *Self-Referencing Spectral Interferometry for Measuring Ultrashort Optical Pulses*, IEEE. J. Quantum. Elect., **35**, 501, (1999).

[145] P. O'Shea, M. Kimmel, X. Gu, and R. Trebino, *Highly Simplified Device for Ultrashort-Pulse Measurement*, Opt. Lett., **26**, 932, (2001).

[146] V. Wong and I. A. Walmsley, *Analysis of Ultrashort Pulse-Shape Measurement Using Linear Interferometers*, Opt. Lett., **19**, 287, (1994).

[147] S. P. Gorza, P. Wasylczyk, and I. A. Walmsley, *Spectral Shearing Interferometry with Spatially Chirped Replicas for Measuring Ultrashort Pulses*, Opt. Exp., **15**, 15168, (2007).

2. Time-Resolved Photoelectron Spectroscopy of Indole and 5-Hydroxyindole

2.1. Introduction

Understanding the way the human body responds to and protects itself from ultraviolet (UV) photoexcitation is important for fundamentally understanding the way we as a species survive. Many researchers have looked at photo-protection mechanisms in DNA [1-4], i.e. how DNA bases and DNA itself rapidly relax after photo-excitation. We are interested in understanding the photo-excitation dynamics in melanin, which acts as our bodies' first line of defence against ultraviolet radiation. In order to understand complex biological molecules, a systematic approach may be taken; by starting with relatively simple molecules and adding groups in a stepwise manner, a fuller picture of the electronic dynamics of large molecules may be gained.

Presented here is a study of the electronic relaxation dynamics of indole and 5-hydroxyindole in the gas-phase, using time-resolved photoelectron spectroscopy (TRPES) in conjunction with *ab-initio* calculations of the geometries and excited state potential surfaces of these molecules. This experimental part of the study was conducted in Albert Stolow's lab in the National Research Council Canada - Steacie Institute for Molecular Sciences. I was based in this lab in Ottawa for two months where I conducted the experiments reported in this chapter with the assistance of Oliver Schalk and others. In addition to gaining experimental data, the experience of working in this lab proved invaluable. This experience was put into good use in designing the software and hardware implemented in the new lab in Heriot-Watt, and running experiments exploring other model biological systems. These new setups and experiments are expanded upon in the next two chapters.

As outlined in Chapter 1, the TRPES approach has a high sensitivity to the dynamics of electronic relaxation in molecular systems and our data offers new insight into the photo-physics of this important class of model biological species. In contrast to previous time-resolved studies, we are able to directly observe the initially prepared ^1L_a excited state decaying on an ultrafast timescale via internal conversion to both the ^1L_b and $^1\pi\sigma^*$ states. We also conclude that the OH group in 5-hydroxyindole plays a minimal role in the relaxation process at the excitation wavelengths studied.

The most common form of melanin in humans is eumelanin which colours the skin, hair and retina and serves to protect the body from the potentially damaging effects of

UV radiation. Eumelanin is a complex polymer made up of three main building blocks; 5,6-dihydroxyindole (DHI), indolequinone (IQ) and 5,6-dihydroxyindole-2-carboxylic acid (DHICA), [5-7] the structures of which are shown in Figure 2.1. It is hoped that more detailed insight into the relaxation dynamics of the electronically excited states that are populated in these different molecules following absorption of UV radiation will help reveal the response of melanin pigments to photoexcitation and the subsequent mechanisms for dissipation of excess energy.

To study the photophysics of melanin in a systematic way, one strategy is to start with a relatively simple model and then gradually incorporate additional substituent groups. As can be seen from Figure 2.1, indole provides a first approximation to all three of the principal constituent units of eumelanin and, as such, it provides a useful starting point for an investigation into electronic relaxation in this important class of biological systems. In addition to the melanin system, indole is the chromophore of the amino acid tryptophan and is also an important sub-unit in many other biological molecules such as the neurotransmitter serotonin and ergot alkaloids [8]. As a next step up in complexity one may then also begin to consider the 5-hydroxyindole molecule. By comparing the relaxation dynamics in these two species, it should be possible to gain insight into the role of the hydroxyl group in the full melanin system.

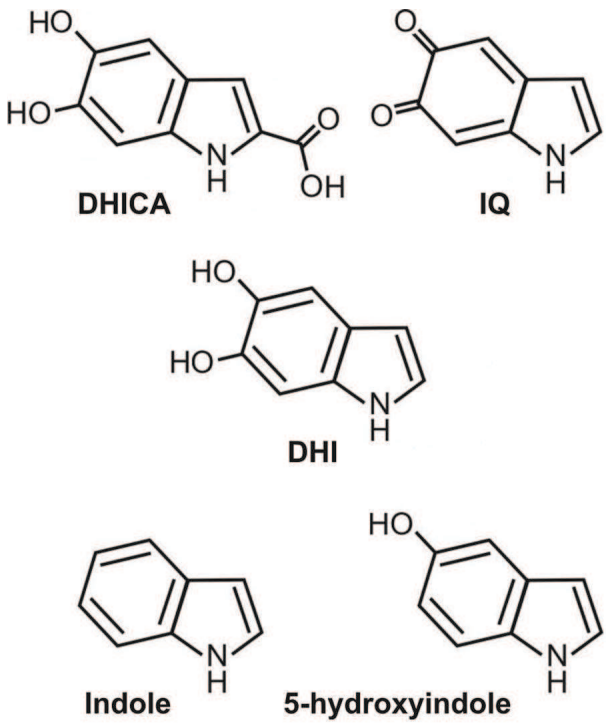


Figure 2.1: Diagrams of the three constituent building blocks of eumelanin: 5,6-dihydroxyindole-2-carboxylic acid (DHICA), indolequinone (IQ) and 5,6-dihydroxyindole(DHI). Also depicted are indole and 5-hydroxyindole, which are the two molecules used in the present study.

The electronic spectroscopy of indole in the gas phase has been well studied and photoexcitation in the 285-220 nm region is primarily to two valence states of $^1\pi\pi^*$ character and $^1A'$ symmetry which are historically labelled ^1L_a and ^1L_b [9]. At energies above ~220 nm, strong absorption to two higher lying $^1\pi\pi^*$

states, denoted ^1B_a and ^1B_b, also becomes significant [10-12]. The region around the ^1L_b origin (which lies at 283.8 nm) has been investigated extensively[13-23] and numerous vibronic bands are observed up to approximately 271 nm. Since no individual vibronic progression exhibits more than two members, the geometries of the S_0 ground state and the ^1L_b state are assumed to be similar. At shorter excitation wavelengths the spectrum becomes increasingly unstructured due to the onset of excitation to the ^1L_a state, the origin of which is believed to be located very close to 273 nm although it has not been observed directly [24, 25].

The lack of observable vibronic structure in the ^1L_a state may be attributed to its short lifetime and this may be rationalised, at least in part, on the basis of results from a recent comprehensive theoretical and experimental study by Schmitt and co-workers [23, 26]. These authors conclude that the ^1L_a state origin is very close to a conical intersection that connects to the ^1L_b state, mediated by Herzberg-Teller active modes (Herzberg-Teller vibrational modes are modes that change the symmetry of a molecule, thus changing the possible allowed vibronic transitions in such a way that the transition dipole moment becomes a function of the inter-nuclear coordinate). This process is essentially barrierless and so vibrationally excited ^1L_a states that are accessed radiatively, funnel directly through into the ^1L_b minimum. Vibronic coupling between the ^1L_a and ^1L_b states (inferred from analysis of the transition dipole moment orientation) also accounts for some of the observed decreases in the ^1L_b state lifetime, which falls from 17.5 ns at the electronic origin to as short as 3-4 ns at around 276 nm. In addition, the anomalously short lifetime of a vibronic band lying just 316 cm^{-1} above the ^1L_b origin, assigned as an out-of-plane mode, was attributed to coupling to an additional, dissociative electronic state of $^1\pi\sigma^*$ character.

Such $^1\pi\sigma^*$ states are now recognised as a common feature in the excited state relaxation dynamics of many molecules containing NH and OH groups, and a number of systems have been extensively studied both experimentally and theoretically over the last decade. The recent review of Ashfold *et al.* (and references therein) provides an excellent starting point for an overview of these investigations [27]. In the case of indole, the $^1\pi\sigma^*$ state has been identified as having $^1A''$ symmetry and significant Rydberg (3s) character in the Frank-Condon region, with much of the electron density being localised on the NH group and exhibiting a node along the N-H bond [4, 28, 29]. The $^1\pi\sigma^* \leftarrow S_0$ transition possesses little or no oscillator strength and so is essentially

"optically dark" to single photon absorption. Calculations from Sobolewski and Domcke using multi-reference *ab initio* methods have suggested that the threshold for internal conversion via a conical intersection connecting the optically bright $^1\pi\pi^*$ states to this $^1\pi\sigma^*$ state lies at around 5 eV (248 nm) above the ground state and is located along the N-H stretching coordinate [4, 29, 30]. The coupling is mediated by out-of-plane vibrational modes of a″ symmetry [31]. These authors also proposed that a second conical intersection, at more extended N-H bond distances, may then give rise to a mechanism for ultrafast internal conversion back to the electronic ground state. This potentially provides a fast non-radiative pathway for the disposal of excess absorbed UV energy. Alternatively, N-H bond fission may occur directly on the $^1\pi\sigma^*$ potential energy surface. The extremely rapid decay of the ^1L_a state via the $^1\pi\sigma^*$ route may also account, in part, for the lack of much observed vibronic structure in the indole absorption spectrum above the ^1L_a origin.

In recent years there have been various experimental studies that have sought to investigate the role of the $^1\pi\sigma^*$ state in indole, although there is a clear lack of consistency in the conclusions that have been reached. Zwier and co-workers recorded fluorescence dip infra-red spectra following UV excitation of several ^1L_b vibronic bands between 283.8 nm and 272.9 nm [32]. Upon observing no significant change in the N-H stretching frequency relative to the ground state, it was concluded that coupling to the $^1\pi\sigma^*$ state does not play a major role in indole at these wavelengths. Evidence of $^1\pi\sigma^*$ involvement was, however, observed in several indole derivatives. This was attributed to the fact that the $^1\pi\sigma^*$ state is highly polar (μ = 11.0 D)[4] and therefore susceptible to large energy shifts relative to the much less polar ^1L_b state (μ = 1.55 D), which may significantly modify the dynamics at similar excitation wavelengths.

Lee and co-workers employed multimass ion imaging techniques to study the photofragment translational energy distribution of the indolyl radical produced following excitation and subsequent dissociation of indole at 248 nm and 193 nm [33]. At both wavelengths they observed a bimodal translational energy distribution and concluded that the high kinetic energy component was the result of direct dissociation via the $^1\pi\sigma^*$ state following internal conversion from the initially prepared $^1\pi\pi^*$ states (which are predominantly ^1L_a at 248 nm and $^1B_a/^1B_b$ at 193.3 nm). The low kinetic energy component was attributed to a "statistical" unimolecular decay following internal conversion to the (highly vibrationally excited) ground electronic state. Similar

findings were also reported by Ashfold and co-workers using H (Rydberg) atom photofragment translational spectroscopy at excitation wavelengths of 193.3 nm and between 240-285 nm [34]. The onset of a structured high kinetic energy H atom elimination channel, attributed to direct dissociation of the $^1\pi\sigma^*$ state along the N-H coordinate, was observed at excitation wavelengths shorter than 263 nm, with low kinetic energy H atoms being assigned to the decay of the indole S_0 ground state. Recent work by Stavros and co-workers employing femtosecond pump-probe spectroscopy also observed a bimodal H atom energy distribution following excitation at 200 nm [35]. These authors noted that the time constant for the appearance of the H atom photofragments was <200 fs in both elimination channels. They assigned the high kinetic energy component to direct dissociation via a $^1\pi\sigma^*$ state following relaxation from the initially excited $^1\pi\pi^*$ state ($^1B_a/^1B_b$) but were unsure of the source of the low kinetic energy signal, as the timescale was significantly shorter than that expected for statistical unimolecular decay from the vibrationally excited ground state.

In contrast to experiments interrogating the photoproducts formed following indole excitation, time-resolved measurements probing the decay of the initially excited states have reached somewhat different conclusions: Radloff and co-workers performed experiments on indole employing femtosecond pump–probe spectroscopy with coincidence detection of photoions and photoelectrons [36]. Their results suggested that, following excitation at 250 nm and 263 nm, the initially prepared $^1\pi\pi^*$ state(s) of indole has a lifetime greater than several hundred picoseconds. Since the short time dynamics that might be expected for rapid internal conversion to the $^1\pi\sigma^*$ state were not observed, it was concluded that no significant coupling to this state was present at these pump wavelengths. Work by Zewail and co-workers using ultrafast electron diffraction also concluded that, following excitation at 267 nm, the $^1\pi\sigma^*$ state does not play a major role in the indole relaxation dynamics [37]. A decaying signal with a time constant of 6.3 ps, was attributed to the initially excited ^1L_a state relaxing via sequential couplings to lower lying $T_2(^3\pi\pi^*)$ and $T_1(^3\pi\pi^*)$ states. These authors also speculated that the $T_1(^3\pi\pi^*)$ state may then be responsible for subsequent H atom loss. Although, given El Sayed's rules,[38] the intersystem crossing (ISC) involved would be expected to take place on a much slower timescale than that observed, the calculated non-planar geometry of the T_2 state was used to argue that vibronic coupling could lead to a significantly increased ISC rate.

In contrast to indole, very little has been reported about the spectroscopy and dynamics of the 5-hydroxyindole molecule in the gas phase. *Ab initio* calculations have determined the two lowest lying optically excited states to be of $^1\pi\sigma^*$ character, with ^1L_a lying above ^1L_b as is the case in indole [39, 40]. The position of the ^1L_b origin sits at 303.9 nm and 305.9 nm for the gauche- and anti- conformers respectively, and at both these excitation wavelengths, as well as in the region between 290-295 nm, the fluorescence lifetime has been reported as 11.1 ns [41-43]. The development of a detailed understanding of the electronic relaxation dynamics of 5-hydroxyindole has, however, very recently taken a major step forward as a consequence of work performed by Ashfold and coworkers [44]. By employing a similar experimental methodology to their previous studies on indole, this group compared the H (Rydberg) atom photofragment translational distributions obtained from hydroxyindole and methoxyindole, substituted at both the 4- and 5- positions, over a range of excitation wavelengths between 303.9 nm and 193.3 nm. Equation of motion coupled cluster theory including single and double excitations (EOM-CCSD) and complete-active-space self-consistent-field (CASSCF) calculations were also employed to assist in interpreting the roles of N-H and O-H bond dissociation in the relaxation dynamics.

In the case of 4-hydroxyindole, O-H bond fission was found to be the dominant source of high kinetic energy H atoms at all excitation wavelengths studied. For 5-hydroxyindole a very different behaviour was observed: high kinetic energy H atoms, originating predominantly from N-H bond fission (as determined by direct comparison with data from 5-methoxyindole) were observed at excitation wavelengths shorter than 255 nm, with some small contributions from O-H dissociation only becoming apparent below 235 nm. Excitation at wavelengths longer that 255 nm produced only low kinetic energy H atom fragments. The significant difference in the photodissociation dynamics exhibited by the 4- and 5- substituted systems was rationalised in terms of their different underlying electronic structures, with the role of the O atom p_x orbital being of particular relevance.

The following study has been published in the Journal of Chemical Physics and builds on the work done by others to provide new insights into the dynamics of indole and 5-hydroxyindole [45].

2.2. Experimental Setup

The gas phase photoelectron studies were conducted in Albert Stolow's lab in Ottawa, Canada. The experimental setup is similar in many ways to the setup described in the next chapter, with the exception of the photoelectron detector, and is described in more detail elsewhere [44]. The molecules of interest, indole (≥99%) and 5-hydroxyindole (97%) were purchased from Sigma-Aldrich and used without any further purification. The solid samples were held in a cartridge mounted inside an Evan Lavie pulsed valve inside the differentially pumped vacuum chamber [46], where they were heated by an external controller. The valve is a high intensity supersonic pulsed molecular beam valve with a 200 μm diameter conical nozzle. Helium at 5 bar pressure was passed over the sample to bring it into the gas phase, and this gas was cooled via supersonic expansion through the valve nozzle. Our choice of carrier gas and expansion conditions means that we do not expect any significant formation of van der Waals complexes [14]. The molecular beam expanded into the source chamber, then passed

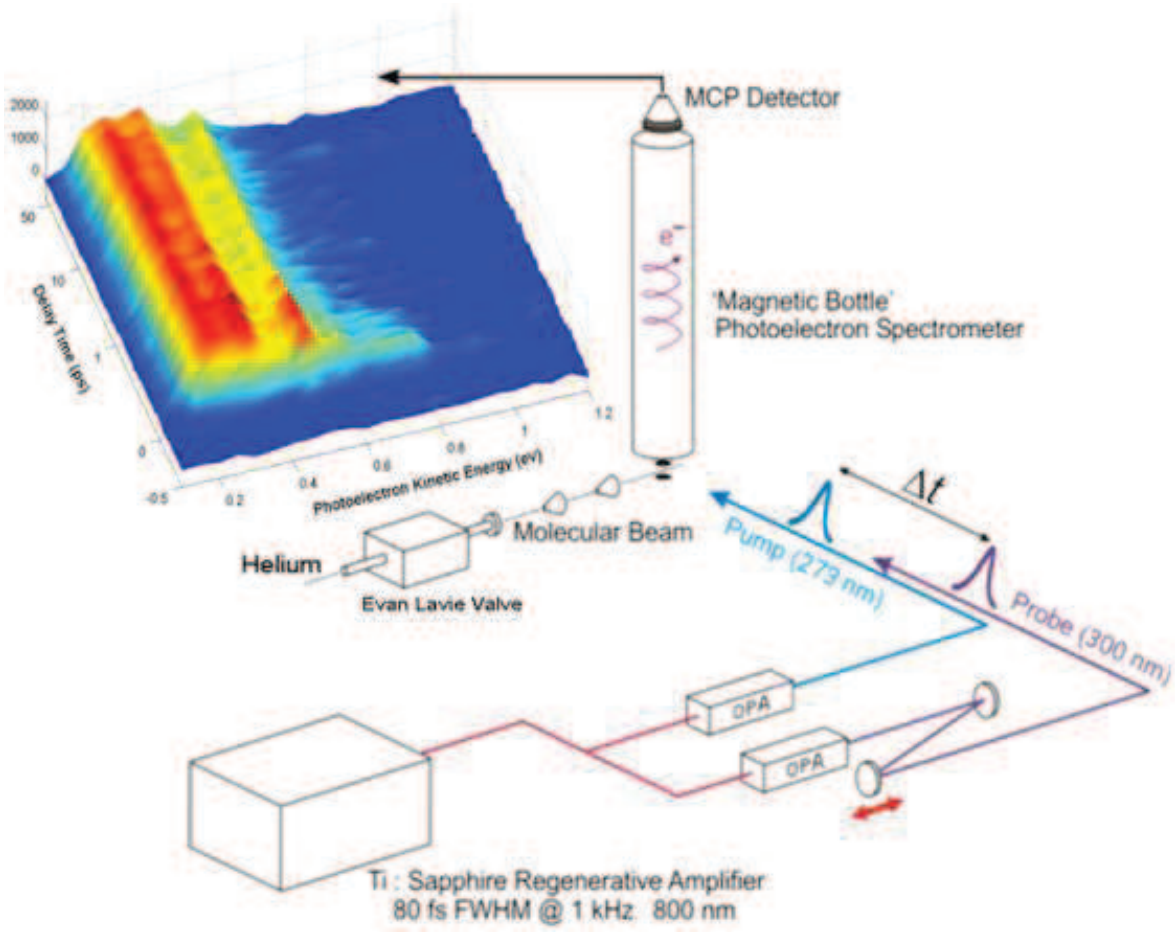


Figure 2.2: Cartoon of the experimental setup in Albert Stolow's group at the NRC Ottawa.

through a skimmer into the interaction chamber, where it was perpendicularly intersected by the co-propagating pump and probe UV pulses. The layout of the experiment can be seen in Figure 2.2. The pump and probe beams were combined on a dichroic mirror and focussed into the interaction region inside the vacuum chamber using a concave spherical aluminium mirror. The focussing conditions of the beams were f/150 for the pump and f/125 for the probe pulse, ensuring a smaller spot size of the probe pulse in the interaction region. The UV pulses were produced using the signal outputs from two identical optical parametric amplifiers (TOPAS, Light Conversion). The 800 nm input for both TOPAS systems was provided by a Ti:Sapphire regenerative amplifier (Positive Light, Spitfire) pumped by two 1 kHz Nd:YLF lasers (Positive Light, Evolution) and seeded by a Ti:Sapphire oscillator (Spectra Physics, Tsunami) pumped by two Nd:YLF diode lasers (Spectra Physics, Millennia). The pump pulses (249 nm or 273 nm) were generated by mixing the signal output of one TOPAS with the 800 nm laser fundamental in a BBO-crystal and subsequently doubling the generated sum frequency. The probe beam (300 nm or 320 nm) was obtained by generating the fourth harmonic of the second TOPAS signal beam output using successive doubling in two BBO crystals. Pulse energies for both pump and probe pulses were attenuated to ~1.3 μJ. The time delay between the pump and the probe was precisely controlled using a computer controlled linear translation stage (Newport, ILS250PP and ESP300). A typical data collection run consisted of stepping the translation stage repeatedly between pump-probe delays of -500 fs to +500 fs in 50 fs steps and 40 exponentially increasing steps between +500 fs and +50 ps. The pump-probe cross correlation width was around 180 fs. This measurement was obtained directly inside the vacuum chamber from non-resonant, two-colour (1 + 1′) multiphoton ionisation of Diazabicyclo[2,2,2]octane (DABCO) when the pump beam was 249 nm, and Azabicyclo[2,2,2]undecane (ABCU) when it was 273 nm.

Photoelectrons were detected by a microchannel plate (MCP) detector positioned at the end of the magnetic bottle flight tube [47, 48]. The magnetic bottle uses a strong inhomogeneous magnetic field (B_i), on the order of one Tesla, to rapidly parallelise all electron trajectories that have any upward motion. The electron moves up the bottle into a weaker constant magnetic field (B_f) of around 10^{-3} Tesla which guides the electrons on a long helical path towards the micro-channel plate detector (see Figure 2.3). At the

micro-channel plate, they produce a chain reaction that results in a small spike in current. The time of flight (t) is related to the energy (E_k) of the electron by:

$$E_k(t) = \frac{m_e\, x^2}{2\, t^2} \tag{2.1}$$

where m_e is the mass of the electron, and $\boldsymbol{x}$ is the length of the flight tube. By comparing the time of the current spikes produced by the MCP plates with the time the laser fires, the time of flight of the charged particles can be calculated. Using this method, almost 50% of the electrons can be detected at a resolution that is approximately equivalent to the femtosecond laser bandwidth, which is approximately 0.02 eV in this experiment.

The magnetic bottle photoelectron data was passed to the computer. At each delay position, the time invariant one-colour pump alone and probe alone background photoelectron signals were dynamically subtracted from the pump-probe signal. Any particularly noisy scans were rejected, and all the data at each time step was summed to give time resolved photoelectron spectra as shown in Figure 2.8 later.

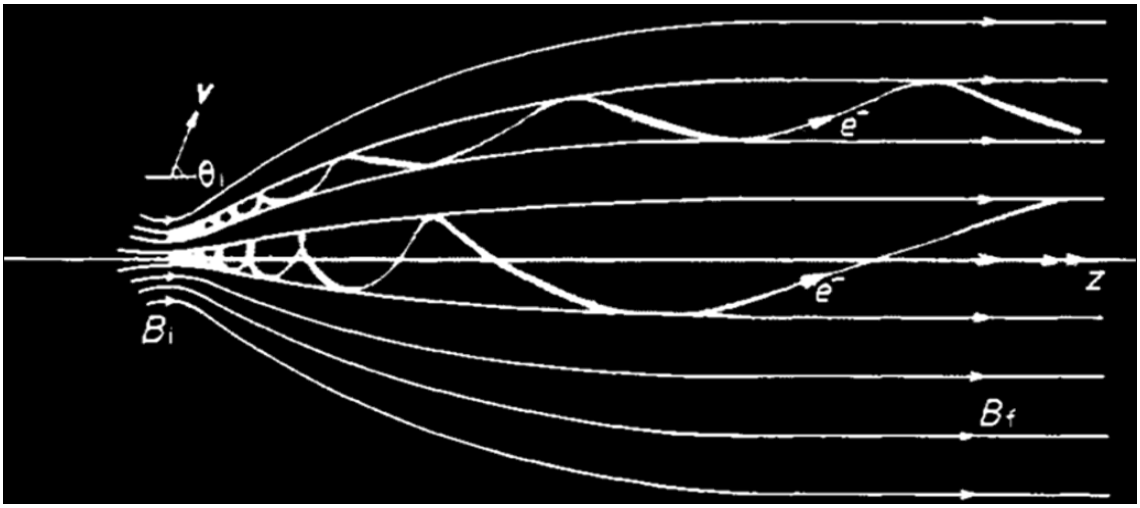


Figure 2.3: Schematic Diagram showing the helical motion of an electron as it moves through the magnetic bottle. Taken from [46].

2.3. Results

2.3.1. Calculations

Calculations of molecular orbitals, potential energy surfaces, and branching space vectors for indole and 5-hydroxyindole were carried out by Therese Bergendahl and Martin Paterson. The ground state geometries of indole and 5-hydroxyindole were optimized by them using density functional theory (B3LYP functional) in conjunction with the aug-cc-pVDZ basis set. Vertical excitation energies and oscillator strengths were calculated using equation of motion coupled cluster theory including single and double excitations (EOM-CCSD) [49]; this is equivalent to linear response (LR) coupled cluster theory for excitation energies. The effect of triples excitations was determined using the CCR(3) method, [50] which gives a non-iterative perturbative correction to LR-CCSD excitation energies, such that excitation energies for singly-excited states are correct through third order in the fluctuation potential. The aug-cc-pVDZ basis was used for all coupled-cluster calculations. GAUSSIAN[51] was used for the B3LYP and EOM-CCSD calculations, while the Dalton program [52] was used for the LR-CCSD and CCR(3) calculations. The carbon and nitrogen core 1s orbitals were frozen in the correlated calculations. The vertical excitation energies and oscillator strengths are shown in Table 2.1. The values obtained reproduce the same state ordering

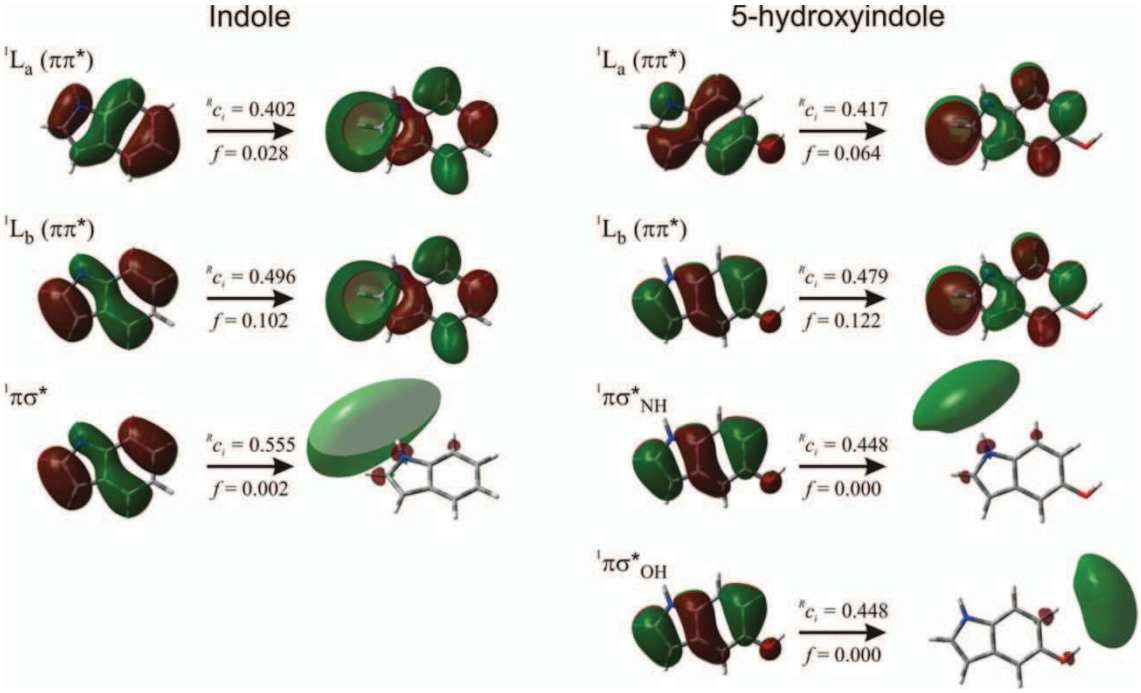


Figure 2.4: Primary orbitals involved in transitions to ^1L_a, ^1L_b and $\pi\sigma^*$ states for indole and 5-hydroxyindole. The magnitude of the EOM-CCSD amplitude, ^Rc_i, is shown. This reflects the importance of this orbital transition to the overall state, as discussed in more detail in the main text. The oscillator strength, f, is also shown for each transition.

as previous calculations and the relative state energies are also in good agreement [30, 44]. The ^1L_b state experiences a significant red-shift upon hydroxylation, in excess of 0.3 eV, whereas the ^1L_a and $\pi\sigma^*_{NH}$ states undergo relatively small changes in energetic position. Figure 2.4 shows qualitative orbital representations of the various excited electronic states. The dominant contribution to the excited state is given by the value of ${}^{R}c_i$, which is the coefficient of the orbital transition in the right hand eigenvector of the EOM state [49]. The orbital transitions shown are the most important contributions to each state in all cases by a factor of at least 5. The ^1L_a transition originates predominantly from the highest occupied molecular orbital (HOMO), while the $^1\pi\sigma^*$ and ^1L_b transitions originate predominantly from the orbital below this (HOMO-1). In indole there is one state of $^1\pi\sigma^*$ character, with electron density localised on the NH group. In 5-hydroxyindole, however, there are two distinct states of $^1\pi\sigma^*$ character with electron density separately localised on the NH and OH groups. Rigid scans along the N-H dissociation coordinates for both molecules and the O-H dissociation coordinate in 5-hydroxyindole were performed and are plotted in Figure 2.5. These were done using EOM-CCSD for the ground state, and the ^1L_a, ^1L_b, and $^1\pi\sigma^*_{NH}$ excited electronic states for indole, and those states plus the $^1\pi\sigma^*_{OH}$ state for 5-hydroxyindole. For the case of indole, EOM-CCSD for N-H dissociation on the $^1\pi\sigma^*$ state predicts a 0.46 eV barrier with the maximum located at 1.4 Å, in good agreement with a recent calculation by Sobolewski and Domcke [30]. A very similar barrier is also present for the $^1\pi\sigma^*_{NH}$ state of 5-hydroxyindole, located at 1.3 Å. For the case of the $^1\pi\sigma^*_{OH}$ state the barrier is

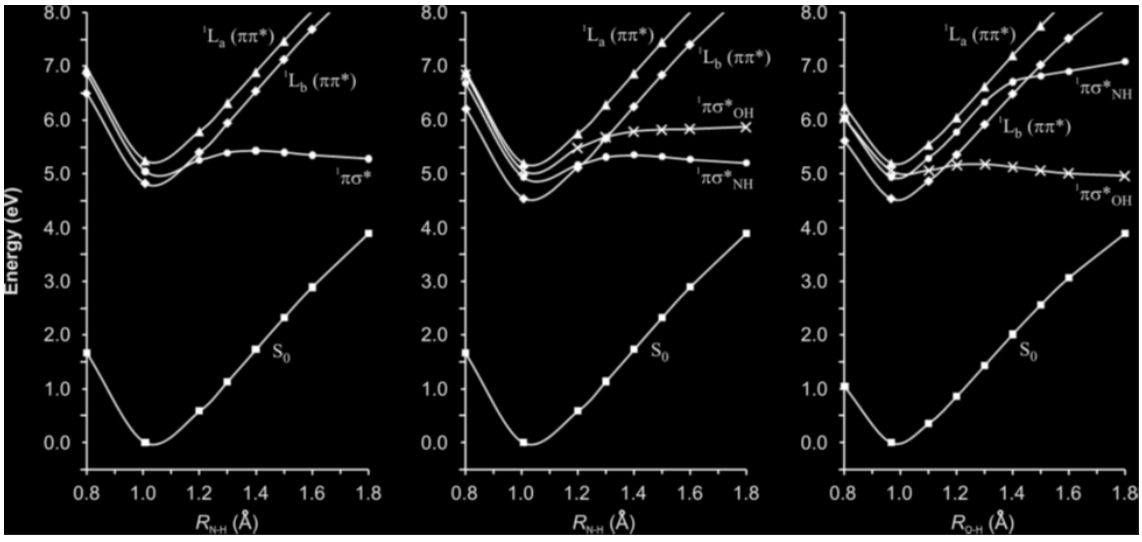


Figure 2.5: Energy cuts, as obtained using EOM-CCSD/aug-cc-pVDZ for indole along the N-H coordinate and 5-hydroxyindole along both the N-H and O-H coordinates.

somewhat smaller, being just in excess of 0.1 eV, and is located at approximately 1.2 Å. These results are also in good agreement with the recent work of Ashfold and co-workers [44]. Complete-active-space self-consistent-field (CASSCF) calculations were also performed, using GAUSSIAN [51], to generate qualitatively correct wavefunctions in regions of strong non-adiabatic coupling. An active space consisting of 12 electrons in 11 orbitals was used, which generated 106953 singlet configurations. The 6-31+G(d) basis set was used for the CASSCF calculations. Conical intersection searches were performed between the S_0 and $^1\pi\sigma^*$ states, the $^1\pi\pi^*$ and $^1\pi\sigma^*$ states, and the two lowest $^1\pi\pi^*$ states for indole. In the CASSCF calculations the two lowest $^1\pi\pi^*$ states correspond to ^1L_a and ^1L_b, however, we note that the relative ordering of these states is highly sensitive to dynamical electron correlation. Moreover, at distorted non-planar geometries the states mix heavily and the ^1L_a and ^1L_b labels lose significance. Orbital rotation derivatives were ignored in the solution to the coupled perturbed multi-configuration self-consistent field (CP-MCSCF) equations, due to the size of the active space. Additionally, symmetry restricted CASSCF was used to optimize the geometry of the $^1\pi\sigma^*$ state. This gave an equilibrium N-H bond length of 1.02 Å. Minimum

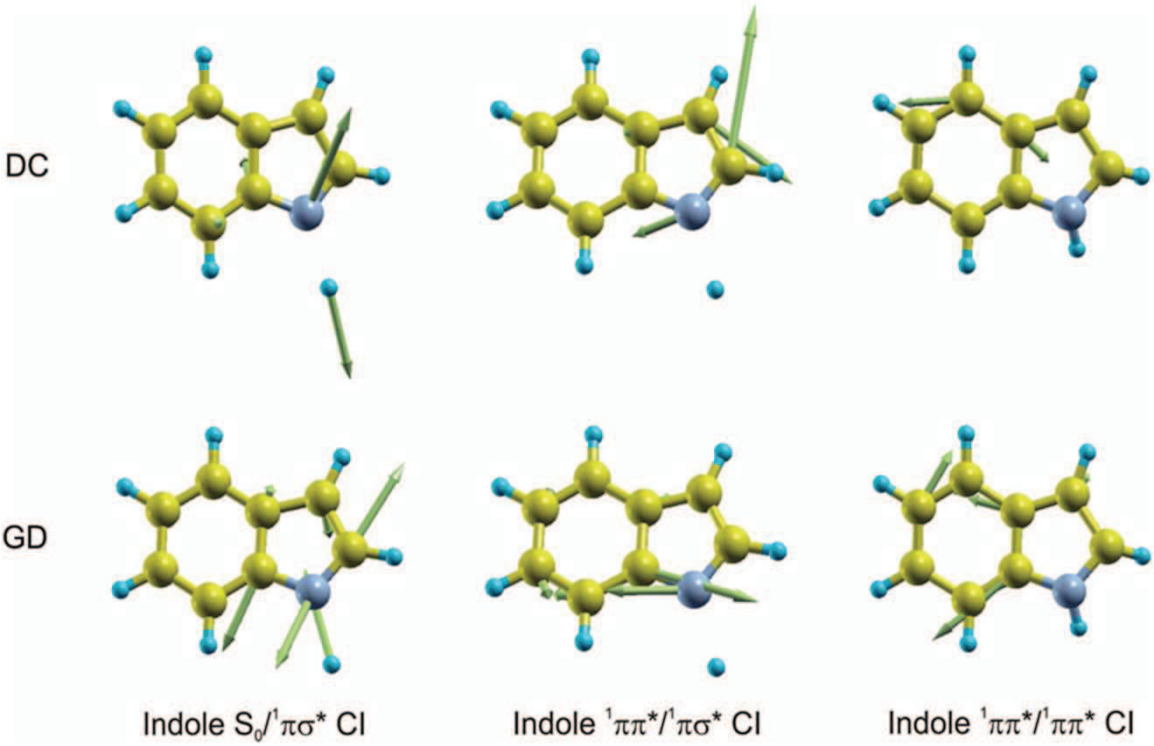


Figure 2.6: Branching space vectors as obtained from CASSCF calculations in indole for the three conical intersections shown. The derivative coupling vector (DC) and the gradient difference vectors (GD) define the directions in which the degeneracy is lifted when moving away from the CI point.

energy crossing points (MECP) in the seam of intersection were obtained for all three pairs of states, and the geometries and branching space vectors for these are shown in Figure 2.6. The branching space defines the two directions in which nuclear motion lifts the degeneracy of the electronic states upon movement through a given conical intersection. In the case of the S_0 and $^1\pi\sigma^*$ states the MECP corresponds to a highly extended N-H bond distance of 1.8 Å, consistent with previous calculations [4, 29, 30]. The branching space here corresponds primarily to motion of the N and H atoms, with some components directed out of the plane of the indole ring. The $^1\pi\pi^*$ and $^1\pi\sigma^*$ states cross at an N-H bond distance of 1.2 Å at the MECP. The branching space is similar to the $S_0/^1\pi\sigma^*$ conical intersection, though it involves a greater in-plane distortion of the pyrrole moiety within the indole ring system. The MECP between the two lowest lying $^1\pi\pi^*$ states involves out-of-plane motions of the N-H and adjacent C-H bonds. The branching space for this conical intersection, however, involves in-plane distortions within the indole ring system. Thus, unlike the previous two conical intersections discussed, the reaction coordinate and branching space are orthogonal. The branching space is broadly similar to that of Schmitt and co-workers as discussed in Refs [23, 26].

State	Indole			5-hydroxyindole		
	LR-CCSD	CCR(3)	f	LR-CCSD	CCR(3)	f
$^1\pi\pi^*$ ^1L_b ($^1A'$)	4.823	4.762	0.0280	4.539	4.458	
$^1\pi\pi^*$ ^1L_a ($^1A'$)	5.241	5.117	0.1018	5.204	5.087	0.1223
$^1\pi\sigma^*$NH ($^1A''$)	5.046	5.017	0.0022	4.945	4.930	0.0000
$^1\pi\sigma^*$OH ($^1A''$)	-	-	-	5.063	5.045	0.0003

Table 2.1: Table showing vertical excitation energies (in eV) for indole and 5-hydroxyindole obtained from coupled cluster response theory in conjunction with an aug-cc-pVDZ one electron basis set. Linear response CCSD (LR-CCSD), equivalent to EOM-CCSD, and CCR(3), which includes non-iterative triples effects are shown. The oscillator strengths, f, are also given, obtained from LR-CCSD.

2.3.2. UV Spectra

UV-Vis absorption spectra of 5-hydroxyindole and indole are shown in Figure 2.7. They were recorded using a Varian 5e-spectrometer, evaporating the samples in a 1 cm long quartz cuvette at 80 ◦C. The spectrum for indole is in good agreement with those published previously [10-12, 53]. Both spectra display a broad peak between 220 nm and 280 nm (which is predominantly due to absorption to the ^1L_a state). It is approximately in the same place in both molecules. At wavelengths longer than 280 nm, however, the narrower peak extending to 295 nm (due to the ^1L_b state) is seen to be substantially broadened and red-shifted in the case of 5-hydroxyindole, with the absorption onset now lying close to 315 nm. The ^1L_b state origin of indole is known to be 283.8 nm [13-17], and is 303.9 nm [42-44] for 5-hydroxyindole. Our spectrum clearly contains significant hot-band contributions, as would be expected given the elevated temperature of the absorption cell used in our measurement. This much larger shift between the two spectra seen in Figure 2.7 in the energetic position of the ^1L_b state compared to ^1L_a is consistent with our calculations (see Table 2.1) and is also in agreement with experimental band decomposition studies carried out on the related 5-methoxyindole system in propylene glycol glass [54] and polyethylene films [55]. The shorter wavelength feature below 220 nm in indole also appears shifted to longer wavelengths and to be slightly broader in 5-hydroxyindole. This is again consistent with observations in 5-methoxyindole, where both the ^1B_b and ^1B_a states are observed to undergo significant red-shifts relative to their position in the indole absorption spectrum [55].

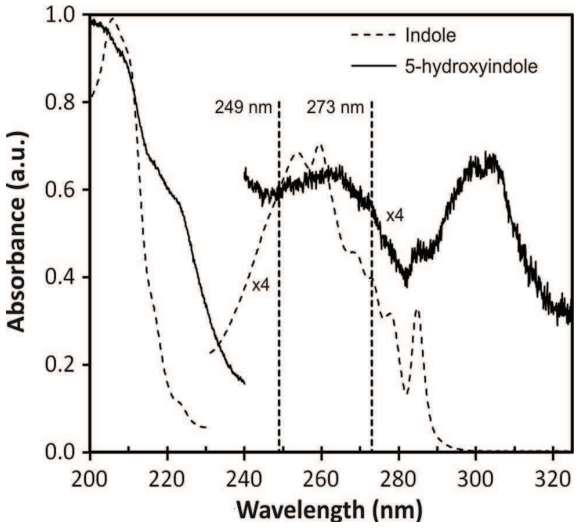


Figure 2.7: Absorption spectra of indole and 5-hydroxyindole in the gas phase. Dashed vertical lines show the excitation energies used in the present study. Samples were evaporated in a 1 cm long quartz cuvette at 80 °C.

2.3.3. Time-Resolved Photoelectron Spectra

Time-resolved photoelectron spectra for both indole and 5-hydroxyindole were obtained at pump wavelengths centred at 273 nm and 249 nm. 5-hydroxyindole was probed at 320 nm, giving combined total (pump + probe) energies of 8.85 eV and 8.42 eV. Indole was probed at 300 nm, which gave combined total energies of 8.67 eV and 9.11 eV respectively. The adiabatic ionisation potential for the D_0 state of the indole cation is 7.76 eV [56-59] and the vertical value is close to 7.9 eV [60-63]. To the best of our knowledge, the only literature data available for the D_0 ionisation potential of 5-hydroxyindole is the vertical value of 7.72 eV reported by Kubota and Kobayashi [63]. The probe wavelength was chosen to give combined total (pump + probe) energies as far above the ionisation potential as possible, and with reference to the molecules absorption spectra (Figure 2.7), to avoid absorption of the probe from the ground state. If the probe was initially absorbed then the results would become complicated due to subsequent ionisation with the pump, leading to dynamics observations on the negative time axis. Figure 2.8 displays a typical data set plotting the photoelectron spectrum as a

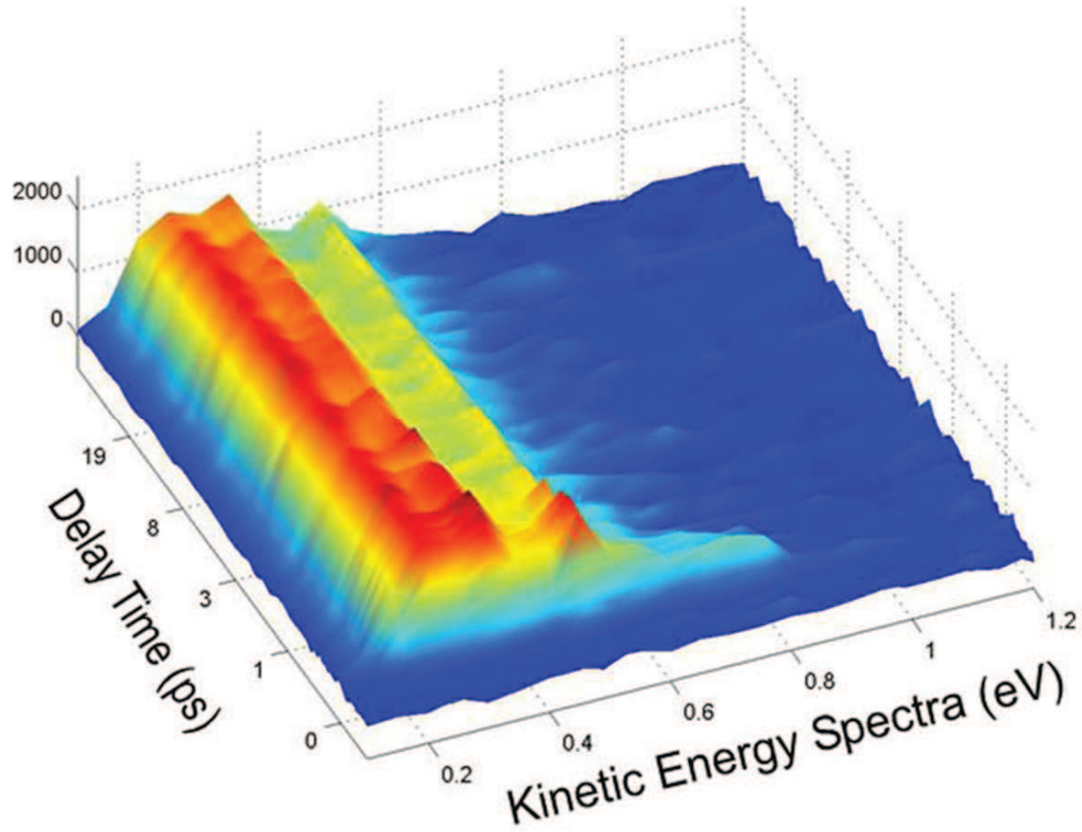


Figure 2.8: Time-resolved photoelectron spectrum of indole following 249 nm excitation and subsequent ionisation with a 300 nm probe. For clarity, the time dependence is plotted using a linear scale in the region -0.5–0.5 ps and a logarithmic scale for delay times between 0.5–50 ps.

function of pump-probe delay. This example shows indole following 249 nm excitation and 300 nm ionisation. For clarity, the time axis is presented on a linear/logarithmic scale, with a linear scale being used for pump-probe delay times less than a picosecond, and a logarithmic scale being used for the longer times. The data is binned into 0.05 eV channels. A very long lived process dominates this spectrum in the 0.0-0.6 eV region, although some decaying signal, superimposed on top of this, is also evident at around 0.5 eV. Additionally, a much more rapidly decaying signal can also be observed at higher kinetic energies between 0.6-1.0 eV. The spectra for indole at 249 nm and for 5-hydroxyindole at both pump wavelengths used all display similar behaviour to that seen in Figure 2.8, although the position of the various features is spectrally shifted to some extent due to the different wavelengths and/or ionisation potential. A more detailed discussion of this will take place in the next section. Time constants and their associated photoelectron spectra were determined using a standard Levenberg-Marquardt global fitting routine wherein the 2D data $S(E, \Delta t)$ are expressed as [64]:

$$I(E,t) = \sum_{i=1-n} A_i(E) \cdot P_i(t) \otimes cc(t) \tag{2.2}$$

Here $A_i(E)$ is the decay associated photoelectron spectrum of the i^{th} channel, which has a time dependent population $P_i(t)$, described by an exponentially decaying function, where n is the number of exponential functions required to non-trivially fit the data, and $cc(t)$ is the experimentally determined Gaussian cross-correlation function. The fitting process is described in more detail in Section 3.3.3. To non-trivially fit all data sets, three exponential functions (all originating from time-zero) were required. One time-constant (τ_1) was shorter than the cross-correlation of the pulses within the machine. For a further discussion on modelling very fast time constants using this method, see the supporting material of Ref. [65]. A second time constant (τ_2) decayed in around a picosecond, and a final time constant (τ_3) was extremely long-lived, effectively modelling a step function over the range of pump-probe delays examined for all data sets with the exception of indole excited at 249 nm. In this case some decay of the long-lived signal was evident and we were able to extract a time constant of 350 ps. Although the margin of error in the fitted time constants was quite large (±20%), the critical advantage of the TRPES approach (when compared to, for example, ion yield experiments which are integrated over all energies) is that the relative amplitude of each exponential component in a given fit may be examined as a *function of electron kinetic*

energy, giving rise to a series of decay associated spectra (DAS) ($A_i(E)$ in Equation 2.2). Deconvoluting the data in this way greatly assists in the interpretation of the overall molecular dynamics since different processes, occurring on very similar timescales, may be *spectrally resolved* from each other, revealing much more information than the time constants alone. Figure 2.9(a-d), displays the DAS obtained from this global multi-exponential fitting process. The evolution of the non-adiabatic relaxation dynamics across different electronic states can be inferred from these DAS and the time constants associated with them. Negative amplitude features are due to the nature of the global fitting procedure we have used. All exponentially decaying functions originate from zero pump-probe delay, and therefore negative amplitudes

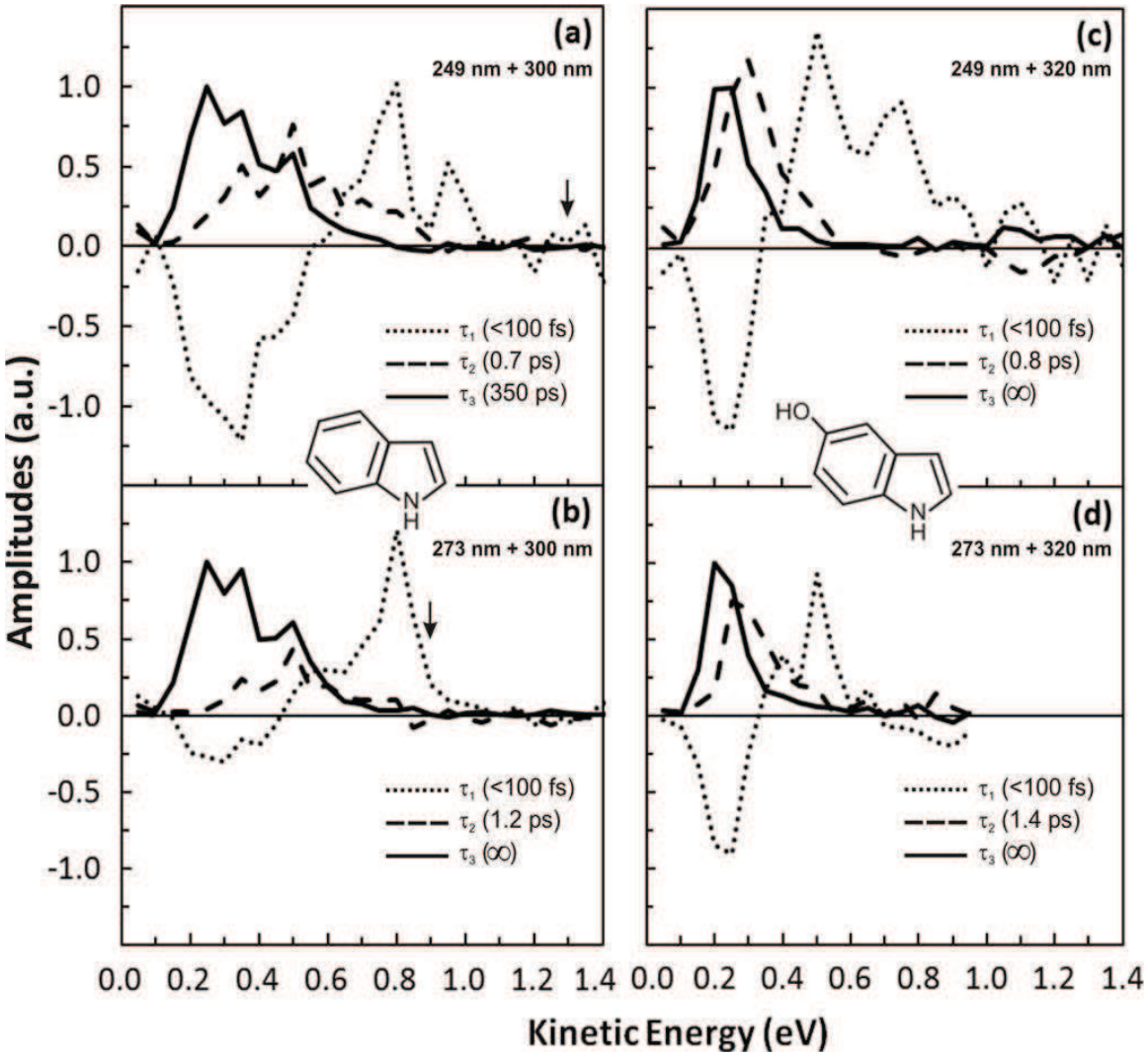


Figure 2.9: Decay associated spectra (DAS) for (a) indole at 249 nm, (b) indole at 273nm, (c) 5-hydroxyindole at 249 nm and (d) 5-hydroxyindole at 273 nm. Numerical time constants $\tau_{1\text{-}3}$ have an uncertainty of ±20%. The individual graphs are normalised with the maximum amplitude of τ_3 set to one, and all other amplitudes then scaled relative to this. The vertical arrows on the indole data denote the maximum photoelectron energy cut-off, based on the known adiabatic ionisation potential of 7.76 eV.

effectively represent an exponential growth in photoelectron signals. These are associated with sequential dynamics, as a new electronic state that was not directly accessed in the initial excitation becomes populated through the decay of the initially excited state. This process is explained in more detail in Section 3.3.3.

2.4. Discussion

2.4.1. Indole at 249 nm

The DAS in Figure 2.9(a) reveal three main processes occurring in the time resolved photoelectron data of indole at 249 nm excitation. We can confidently assume that we are exciting almost exclusively to the ^1L_a state, because of several band decomposition studies in the UV absorption region that have been carried out on crystalline indole [66] as well as indole in polymer films [67] and propylene glycol glass [54, 68, 69]. Even given the small shifts in the relative and absolute positions of the ^1L_a and ^1L_b states in these solid media relative to the gas phase, our excitation wavelength should only excite the ^1L_a state. Because of this, we are able to attribute the short-lived (<100 fs) positive feature between 0.6-1.0 eV in the DAS to the rapidly decaying ^1L_a state.

The negative amplitude feature in the τ_1 DAS between 0.1-0.6 eV represents the growth of population in new electronic states following electronic relaxation. At low photoelectron kinetic energies (< 0.25 eV), the shape of the feature, as well as the relative amplitude and position is well matched to that of the positive DAS for the long lived feature with time constant τ_3 = 350 ps. This would suggest that the initially excited ^1L_a state is decaying into this longer lived state.

Glasser and Lami performed measurements of the non-radiative, collision-free decay rates in indole. At the excitation wavelength we are using, their data suggests that the lifetime is approximately 180 ps, which is within a factor of 2 of our time constant τ_3 [70]. It should be noted here that this work also speculated that the decrease in lifetime with increasing excitation energy was due to the ^1L_a state having electron density localized on the NH group and undergoing efficient N-H bond fission, effectively predicting the non-radiative decay mechanism which we now attribute to the $^1\pi\sigma^*$ state (see below). All the most recent calculations, including those by Sobolewski and Domcke [4, 29, 30] and ourselves, conclude that the ^1L_a state is strongly bound along the N-H coordinate. In addition, Schmitt and co-workers [26] predict a conical

intersection between the ^1L_a and ^1L_b states. With this in mind, we suggest that at this wavelength the τ_3 DAS comes from the long lived ^1L_b state, populated through rapid internal conversion from the initially excited ^1L_a state.

We clearly see ultrafast dynamics in our indole data, in contrast to the findings of Radloff and co-workers [36], whose time-resolved photoelectron spectroscopy studies observed only long lived results at pump wavelengths of 250 nm and 263 nm. The reason for this discrepancy is not clear, although we speculate that it may be due to differences in the choice of probe scheme used in the two experiments. We use a one-photon 300 nm probe scheme in contrast to the Radloff study that used predominantly two-photon ionisation at 400 nm. The different schemes will have differences in Franck-Condon factors and ionisation cross-sections, and so may be able to detect different states.

The agreement between the τ_3 signal and the negative amplitude region of the τ_1 DAS is clearly less good at photoelectron kinetic energies above 0.25 eV. This may suggest that a second electronic state may also be populated following internal conversion from ^1L_a and that this also influences the negative amplitude τ_1 DAS signal. The $^1\pi\sigma^*$ state that has been previously implicated experimentally and theoretically in providing a direct dissociation pathway that leads to H atom photofragments with high kinetic energies would seem to be the obvious candidate. H atom fragments from this state at excitation wavelengths shorter than 263 nm have previously been observed by Ashfold and co-workers [34]. We suggest that population of this $^1\pi\sigma^*$ state, following internal conversion from ^1L_a, is also partly responsible for the observed negative shape of the τ_1 DAS above 0.25 eV.

The $^1\pi\sigma^*$ state is essentially optically dark to single photon absorption, and as such it is not accessed directly to any significant extent in the initial excitation but must obtain population via non-adiabatic coupling. We therefore also take the τ_2 DAS to be a direct signature of $^1\pi\sigma^*$ involvement and suggest that this is a competing mechanism with relaxation via ^1L_b. We are unable to definitively determine the branching ratio for these two pathways since we do not know the relative ionisation cross sections of the $^1\pi\sigma^*$ and ^1L_b states. The $^1\pi\sigma^*$ lifetime of 0.7 ps would appear to suggest that we are sitting just above the 0.45 eV barrier that lies along the N-H coordinate outside the Franck-Condon region (see Figure 2.5). Other similar systems such as pyrrole have a much shorter $^1\pi\sigma^*$ lifetime (<100 fs) [71], however the previously mentioned study by

Ashfold and co-workers[34] may also suggest that the indole $^1\pi\sigma^*$ lifetime is longer than usual. It was noted by this group that the shape and intensity of the H atom kinetic energy release spectra they obtained were insensitive to the relative alignment of the photolysis laser polarization, implying an essentially isotropic distribution of fragment recoil velocities. This isotropy was true for all fragments, including the fast fragments arising from direct N-H dissociation on the $^1\pi\sigma^*$ surface, which were only observed at excitation wavelengths shorter than 263 nm. On the assumption that the excitation is mainly to a single electronic state (which, as discussed earlier, we take to be the case for our data at 249 nm), this may indicate that the $^1\pi\sigma^*$ state lives long enough for the initially aligned indole molecules to de-phase to some extent. This data is in contrast to the more rapidly decaying pyrrole, where strongly anisotropic recoil of fast H-atom fragments has been observed [72, 73].

Following the population of the $^1\pi\sigma^*$ state at 249 nm the indole molecule has been observed to undergo dissociation along the N-H bond. This yields an H atom and the ground state indolyl radical. This has been inferred from the previously discussed fast kinetic energy channels reported in studies probing either the indolyl or H atom photoproducts [33, 34]. A second possible, alternative route is non-radiative relaxation to indole's vibrationally excited ground electronic state. This could lead to "statistical" fragmentation to give H atoms in conjunction with electronically excited indolyl radicals. This mechanism is thought to be responsible for the slow kinetic energy H atom channel in the frequency resolved measurements of photoproducts that have reported to date. In indole this may occur via the proposed conical intersection between the $^1\pi\sigma^*$ and S_0 states at extended N-H bond distances. However, as will become apparent when we come to consider our data at 273 nm, we cannot rule out the possibility of other, additional pathways.

Another potential outcome for the $^1\pi\sigma^*$ state could involve internal conversion to the ^1L_b state. However, since we are able to ionise, and thus observe the ^1L_b state, if the ^1L_b state were being populated from the $^1\pi\sigma^*$state we might expect to see some form of signal appearing in the photoelectron spectrum with a rise time that matched the $^1\pi\sigma^*$ decay lifetime. The fact that we see no such signals leads us to conclude that this pathway is not of major significance, although we note that, in the event of the $^1\pi\sigma^*$ state possessing a very much larger ionisation cross section than the ^1L_b state, the effect may be so small that we cannot detect it. The lack of a rising signal anywhere in the

data that is matched to the decay of the τ_2 DAS also leads us to rule out the possibility of the τ_2 DAS being due to the second step in a bi-exponential decay of the ^1L_a state (that then relaxes exclusively to ^1L_b), rather than the $^1\pi\sigma^*$ state.

The ultimate fate of the ^1L_b state still remains unclear. We observe it to decay on a timescale of several hundred picoseconds following rapid internal conversion from ^1L_a. The fluorescence quantum yield for indole's ^1L_b state is relatively low ($\phi \sim 0.1$) [70], and so there must be a non-radiative decay pathway for this state that has not yet been characterised. It could be speculated that this may be an additional source of the "statistical" H atoms that have been previously observed in the frequency resolved studies. The time-resolved electron diffraction data of Zewail and co-workers [37] suggested that a triplet state is accessed via intersystem crossing from the initially excited state following excitation at 267 nm. They observed a rise time of 6.3 ps for the population of a long lived state. Our measurements show no obvious evidence of this; our TRPES data exhibit no signals with comparable dynamical timescales. Even if our experiment was effectively "blind" to the triplet state itself due to an unfavourable ionisation cross section, we suggest that observation of a decaying signal in some region of the photoelectron spectrum might be expected due to disappearing population in the state from which any triplet state is being populated.

We now turn our attention back the τ_1 DAS in Figure 2.9(a). Unlike the DAS for τ_2 and τ_3, which we take to be direct representations of the photoelectron spectra associated with ionisation from the $^1\pi\sigma^*$ state and ^1L_b state respectively, the τ_1 DAS is not a true representation of the photoelectron spectrum associated with the ^1L_a state. This is due to the nature of the global fitting procedure we have used to analyse the data, which assumes that all exponentially decaying signals originate from zero pump-probe delay. In order to fit photoelectron signals that grow in after some time,

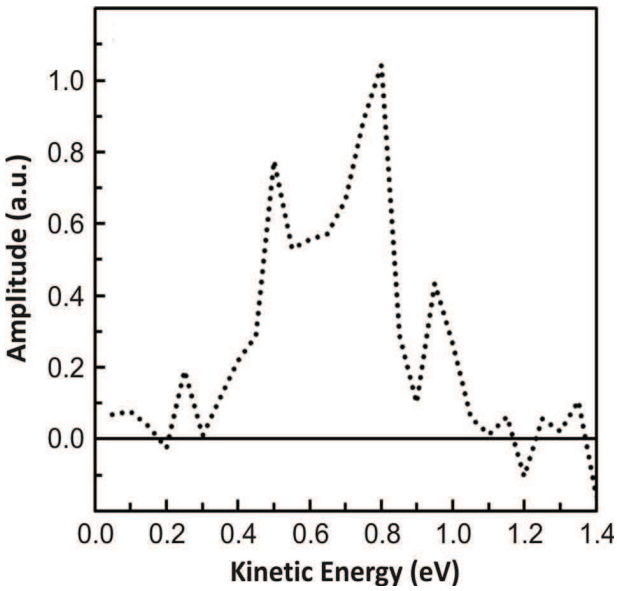


Figure 2.10: Sum of the three individual decay associated spectra (DAS) presented in Figure 7(a) for indole excited at 249 nm. For more details, see the main text.

the fit introduces negative amplitudes into the DAS. The τ_1 DAS exhibits negative amplitudes in order to compensate for photoelectron signals that originate from electronic states populated following internal conversion after the initial excitation. Since we have determined that the population of both the ^1L_b and $^1\pi\sigma^*$ states originates directly from the initially prepared ^1L_a state we are, in this particular instance, able to extract the true ^1L_a photoelectron spectrum by adding together the DAS for the three individual time constants used in the fit. By this method we are able to isolate the component of the τ_1 DAS associated exclusively with the decay of the ^1L_a state, due to the previously discussed assumption that there is no significant excitation directly to the ^1L_b state at this wavelength excitation.

The result of this operation is shown in Figure 2.10. In addition to the peaks at 0.8 eV and 0.95 eV that were visible in the τ_1 DAS in Figure 2.9(a) shown on page 48, a new feature at 0.5 eV, that was previously obscured, is now also readily apparent. At 249 nm excitation, we are exciting to 0.44 eV above the proposed ^1L_a origin, and the total (pump + probe) energy available is 9.11 eV. The ^1L_a state is expected to ionize predominantly into the D_0 state of the indole cation as it originates from the HOMO. Taking the vertical ionisation potential of 7.9 eV reported in the He(I) photoelectron spectrum of indole [60-63], we would therefore predict a peak corresponding to diagonal ionisation (i.e. $\Delta v = 0$, where v is a generalised, non-mode-specific quantum number) to appear at close to 0.8 eV in the τ_1 DAS. Indeed, this is what we observe.

The smaller features at 0.95 eV and 0.5 eV also provide some indication of ionisation events for which $\Delta v \neq 0$, although the absence of a long progression extending all the way to the maximum photoelectron energy cut off (which is 1.35 eV in this instance) would suggest no major changes in geometry upon ionisation. Additionally, in the case of ionisation from the ^1L_b state, the molecule is sitting at 0.61 eV above the ^1L_b origin following ^1L_a excitation at 249 nm and subsequent internal conversion. Given that the ^1L_b state originates from the HOMO–1 orbital, diagonal ionisation into the D_1 state of the indole cation (which is known to lie at 8.36 eV [60-63]) would therefore be expected to give rise to a strong feature close to 0.15 eV in the τ_3 DAS. We do not see this in our data, although we note that the transmission efficiency of our spectrometer decreases sharply at very low photoelectron kinetic energies. Due to this we are unable to comment further on the nature of the ^1L_b ionisation.

2.4.2. Indole at 273 nm

Figure 2.9(b) shows the three DAS obtained for indole at the longer excitation wavelength of 273 nm. The two data sets for indole generally appear very similar, and it is interesting to note that the spectral features in all three DAS appear at the same photoelectron kinetic energies for both excitation wavelengths. This suggests that the nature of the ionisation event is similar for both cases, and that the extra 0.44 eV imparted by the shorter wavelength is directly mapped into the indole cation. The total photon (pump + probe) energy available in this case is 8.67 eV and the vertical ionisation potential is 7.9 eV. From this we would predict the $\Delta v = 0$ highest energy photoelectron peak to appear at close 0.8 eV in the τ_1 DAS, as is observed in Figure 2.9(b). The $\Delta v < 0$ peak at 0.95 eV in the 249 nm data is not present at 273 nm as we are exciting at the ^1L_a origin. The spectral positions of all other features in the three DAS at 273 nm are essentially identical to those observed at 249 nm.

There are several additional important points to draw from a comparison of Figure 2.9(a) and (b). Firstly, at 273 nm we still see a clear signature of decay via the $^1\pi\sigma^*$ state, which we have attributed to the τ_2 DAS, providing direct evidence that this state is involved in the relaxation dynamics at longer wavelengths than the 263 nm threshold for the production of fast H atom fragments observed by Ashfold and co-workers [34]. This observation represents a good example of the complementary nature of time- and frequency-resolved measurements. The observation of fast H atom fragments at wavelengths shorter than 263 nm allows us to definitively assign the τ_2 DAS in our data at 249 nm to the $^1\pi\sigma^*$ state. The appearance of this same τ_2 DAS at 273 nm then allows us to subsequently assign a role to the $^1\pi\sigma^*$ state in the indole relaxation dynamics below the fast H-atom threshold.

Secondly, the time constant τ_2 is longer at 273 nm than at 249 nm (1.2 ps *vs*. 0.7 ps). This is significant as the magnitude of this increase is larger then anything that can simply be attributed to the uncertainty of our fitting procedure, although the timescale of the decay is still sufficiently short to suggest that tunnelling through the barrier along the N-H coordinate is unlikely to be a prevailing pathway. This is particularly apparent given that the reduction in excitation energy in moving from 249 nm excitation wavelength to 273 nm is 0.44 eV, and the decay rate is less than a factor of 2 slower. The $^1\pi\sigma^*$ state is therefore unable to decay via internal conversion to the S_0 state along

the N-H coordinate as the relevant conical intersection lies at an extended distance, on the other side of the barrier (see Figure 2.5). An alternative decay pathway, mediated by a different vibrational mode, must therefore be responsible. In order to investigate this possibility further, we performed constrained conical intersection searches at a variety of restricted N-H distances. We were unable to find any points of degeneracy, however, between the $^1\pi\sigma^*$ state and the S_0 state under these conditions and so the nature of any alternative radiationless decay pathway at short N-H distances remains an open question.

An additional point of note in Figure 2.9(b) is that the relative amplitudes of the features associated with the three time constants are clearly different to those observed at 249 nm. In particular, the negative amplitude feature in the τ_1 DAS is significantly smaller following excitation at 273 nm. Although this could, in part, be a consequence of the different Franck-Condon factors involved in the ionisation step following excitation at the two different wavelengths, at around 273 nm, the indole fluorescence excitation spectrum exhibits a well resolved vibronic band structure characteristic of direct excitation to a long lived state. Data reported by several groups shows a cluster of several sharp peaks in the region between 272 – 274 nm and these are thought to be ^1L_b vibronic states that derive their intensity through Herzberg-Teller vibronic coupling with ^1L_a [16, 24, 25]. The broad bandwidth (~150 cm^{-1} FWHM) of the ultrafast pulses used in our experiments will excite all of these states simultaneously and, as such, some contribution to the signal described by the τ_3 DAS will now be from direct excitation to these ^1L_b state vibronic bands and will therefore originate from zero pump-probe delay.

Finally, we are unable to extract a decay time for the τ_3 DAS at 273 nm, in contrast to the situation at 249 nm excitation, as it is essentially a step function over the range of pump-probe delays employed (Δt_{max} = 50 ps). This is consistent with Glasser and Lami's observations [70], which suggests a lifetime for the ^1L_b state on the order of several nanoseconds for indole excited close to the proposed ^1L_a origin.

2.4.3. 5-Hydroxyindole at 249 nm and 273 nm

The DAS for 5-hydroxyindole, as shown in Figure 2.9(c) and (d), appears to display broadly similar dynamics to those previously discussed for indole. Both the spectral content and the related timescales seem to be in broad agreement. Due to a lower probe energy, the resultant photoelectron energies are shifted slightly, as would be expected. Due to the similarities, we can conclude that 5-hydroxyindole exhibits similar electronic relaxation mechanisms to those observed in indole. We assume that non-adiabatic coupling occurs between an initially excited ^1L_a state and both the ^1L_b and $^1\pi\sigma^*$ states. 5-hydroxyindole has two $^1\pi\sigma^*$ states, dissociative along the N-H and O-H bonds respectively. This gives two channels for H atom elimination or internal conversion back to the ground state.

At a purely heuristic level, it would seem reasonable to assume that the dynamical timescales and/or relative amplitudes and spectral features of the DAS we obtain for 5-hydroxyindole at a given excitation wavelength should be significantly altered, relative to the case of indole, if the additional O-H pathway was active and populated. As there are no significant differences between the DAS we are led to the conclusion that the addition of the OH group to the indole molecule in this position has little effect and the OH group does not play a significant role in the dynamics at the excitation wavelengths studied. This is consistent with the recent findings of Ashfold and co-workers [44], who have suggested that the OH group in 5-hydroxyindole is a spectator in the relaxation process due to the nature of the interaction between the O atom p_x orbital and the π system of the indole ring, with the relative destabilizing contributions of this orbital to the HOMO and HOMO-1 being the key consideration.

Although the major point of our argument rests on the similarity between the DAS of indole and 5-hydroxyindole, there are some minor differences that reveal interesting points. Firstly, in contrast to indole, we do not see such a dramatic reduction in the amplitude of the negative portion of the τ_1 DAS in 5-hydroxyindole at 273 nm excitation when compared to 249 nm. As discussed in Section 2.3.2 the ^1L_b state is significantly red shifted with the addition of the OH group in 5-hydroxyindole. This would lead to less direct excitation to this state at 273 nm. As such, a smaller signal level in the τ_3 DAS (associated with ionisation of the ^1L_b state) will therefore originate from zero pump-probe delay. Secondly, at both excitation wavelengths used in our study we were unable to extract a time constant τ_3 for 5-hydroxyindole given the limited

(≤ 50 ps) range of pump-probe delays that were sampled. Both appeared to act as a step function with no appreciable decay over this time period. Once again turning to the non-radiative lifetime study of Glasser and Lami,[70] we note that this would seem to be broadly consistent with the observation of lifetimes on the order of 1 ns that were reported for the related 5-methoxyindole species at both 273 nm and 249 nm. Finally, Sobolewski and Domcke have published an interesting theoretical dynamical study on the photochemistry of the related 5,6-dihydroxyindole molecule. These authors proposed that the initially excited $^1\pi\pi^*$ state(s) (^1L_a and ^1L_b) may relax non-adiabatically via the $^1\pi\sigma^*_{OH}$ state leading to hydrogen migration to the neighbouring carbon atom of the six-membered ring [74] to form a new structure denoted 6-hydroxy-4-dihydro-indol-5-one (HHI). This structure is predicted to absorb strongly in the visible region of the spectrum, undergoing a rapid transition to an excited state of $^1\pi\sigma^*$ character which then relaxes back to the HHI ground state on an ultrafast timescale. It was suggested that this overall process may play an important role in the photoprotection provided by the eumelanin system. In principle, a similar mechanism might also be possible in 5-hydroxyindole, although we note that the HHI system may receive additional stabilization from the interaction between the two oxygen atoms (via the H atom of the OH group attached to the 6 position) that would obviously be absent in the mono-hydroxylated system. The stabilisation effect of two OH groups on an aromatic ring has been studied in detail, and is the focus of Chapter 5. The migration of an H-atom onto the 6-membered ring would induce significant changes in the geometry of the molecular framework and this should be dramatically reflected in the appearance of the photoelectron spectra we see. The very similar nature of the data we have obtained for the two molecules leads us to suggest that, at the excitation wavelengths we have employed, this H atom migration mechanism does not play a significant role in 5-hydroxyindole's photodynamics.

2.5. Conclusions

We have investigated the non-adiabatic relaxation dynamics of indole and 5-hydroxyindole at excitation wavelengths of 249 nm and 273 nm using time-resolved photoelectron spectroscopy (TRPES). Indole and 5-hydroxyindole exhibit similar dynamics at the wavelengths used in this study. Initial excitation is primarily to the ^1L_a state, which decays quickly (< 100 fs) through conical intersections to the lower lying ^1L_b and $^1\pi\sigma^*$ states. This is in contrast to some previous time-resolved studies performed on indole where the involvement of the $^1\pi\sigma^*$ state has not been observed and the $^1L_a/^1L_b$ relaxation dynamics were not deconvoluted. The ^1L_b state has a very long lifetime within the timescale of the experiment, ultimately relaxing to the ground state through fluorescence [38]. The $^1\pi\sigma^*$ states are located along the N-H bond in indole, and along both the N-H and O-H bonds in 5-hydroxyindole, and decay more rapidly at higher energy excitation. The involvement of the $^1\pi\sigma^*$ state in the dynamics at 249 nm is consistent with the data from frequency-resolved experiments studying the kinetic energy release distribution of photoproducts. We are able to directly observe the role of the $^1\pi\sigma^*$ state at considerably longer wavelengths than has been observed previously. The similarity of the spectrally resolved dynamics observed in 5-hydroxyindole and indole at both excitation wavelengths leads us to conclude that the role of the OH group in 5-hydroxyindole is minimal. This observation is also consistent with very recent frequency-resolved studies [44].

In addition to gaining valuable information about the photodynamics of indole and 5-hydroxyindole, the experience gained through setting up these experiments, collecting and analysing the data, as well as extensive discussions with the staff and students at the National Research Council Canada, was invaluable in setting up the new lab in Heriot-Watt University. Observing how their software interfaced with the hardware, and what features were most used and most helpful in the setup of experiments and analysis of data allowed me to make my software both more user friendly and more flexible than it was before. My involvement with both the optical and vacuum components of the experiment setup gave me insight into the everyday running of the equipment that was very helpful in designing similar equipment in Scotland that should be useful many years after I have left, as discussed in the next chapter.

2.6. References

[1] H. Fernando, G. A. Papadantonakis, N. S. Kim, and P. R. LeBreton, *Conduction-Band-Edge Ionization Thresholds of DNA Components in Aqueous Solution*, P. Natl. Acad. Sci. USA, **95**, 5550, (1998).

[2] J. Cerny, V. Spirko, M. Mons, P. Hobza, and D. Nachtigallova, *Theoretical Study of the Ground and Excited States of 7-Methyl Guanine and 9-Methyl Guanine: Comparison with Experiment*, Phys. Chem. Chem. Phys., **8**, 3059, (2006).

[3] M. N. R. Ashfold, B. Cronin, A. L. Devine, R. N. Dixon, and M. G. D. Nix, *The Role of $\pi\sigma^*$ Excited States in the Photodissociation of Heteroaromatic Molecules*, Science, **312**, 1637, (2006).

[4] A. L. Sobolewski, W. Domcke, C. Dedonder-Lardeux, and C. Jouvet, *Excited-State Hydrogen Detachment and Hydrogen Transfer Driven by Repulsive $^1\pi\sigma^*$ States: A New Paradigm for Nonradiative Decay in Aromatic Biomolecules*, Phys. Chem. Chem. Phys., **4**, 1093, (2002).

[5] P. Meredith and T. Sarna, *The Physical and Chemical Properties of Eumelanin*, Pigm. Cell. Res., **19**, 572, (2006).

[6] P. Meredith, B. J. Powell, J. Riesz, S. P. Nighswander-Rempel, M. R. Pederson, and E. G. Moore, *Towards Structure-Property-Function Relationships for Eumelanin*, Soft Matter, **2**, 37, (2006).

[7] A. Huijser, A. Pezzella, and V. Sundstrom, *Functionality of Epidermal Melanin Pigments: Current Knowledge on UV-Dissipative Mechanisms and Research Perspectives*, Phys. Chem. Chem. Phys., **13**, 9119, (2011).

[8] F. R. D. Alves, E. J. Barreiro, and C. A. M. Fraga, *From Nature to Drug Discovery: The Indole Scaffold as a 'Privileged Structure'*, Mini-Rev. Med. Chem., **9**, 782, (2009).

[9] J. R. Platt, *Classification of Spectra of Cata-Condensed Hydrocarbons*, J. Chem. Phys., **17**, 484, (1949).

[10] H. Lami, *Possible Role of a Mixed Valence-Rydberg State in Fluorescence of Indoles*, J. Chem. Phys., **67**, 3274, (1977).

[11] H. Lami, *Presence of a Low-Lying Rydberg Band in Vapor Absorption-Spectra of Indole and 1-Methyl Indole*, Chem. Phys. Lett., **48**, 447, (1977).

[12] P. Ilich, *Lowest Singlet-States in Isolated Indoles*, Can. J. Spectrosc., **32**, 19, (1987).

[13] J. M. Hollas, *Vapour-Phase Ultra-Violet Absorption Spectra of Indene, Indole, Coumarone and Thionaphthene*, Spectrochim. Acta., **19**, 753, (1963).

[14] J. Hager and S. C. Wallace, *Laser Spectroscopy and Photodynamics of Indole and Indole-Vanderwaals Molecules in a Supersonic Beam*, J. Phys. Chem., **87**, 2121, (1983).

[15] Y. Nibu, H. Abe, N. Mikami, and M. Ito, *Electronic-Spectra of Hydrogen-Bonded Indoles in a Supersonic Free Jet*, J. Phys. Chem., **87**, 3898, (1983).

[16] R. Bersohn, U. Even, and J. Jortner, *Fluorescence Excitation-Spectra of Indole, 3-Methyl Indole, and 3-Indole Acetic-Acid In Supersonic Jets*, J. Chem. Phys., **80**, 1050, (1984).

[17] J. W. Hager, D. R. Demmer, and S. C. Wallace, *Electronic-Spectra of Jet-Cooled Indoles - Evidence for the ^1L_a State*, J. Phys. Chem., **91**, 1375, (1987).

[18] G. A. Bickel, D. R. Demmer, E. A. Outhouse, and S. C. Wallace, *The S_1-S_0 Transition of Indole and N-Deuterated Indole - Spectroscopy and Picosecond Dynamics in the Excited-State*, J. Chem. Phys., **91**, 6013, (1989).

[19] D. M. Sammeth, S. X. Yan, L. H. Spangler, and P. R. Callis, *2-Photon Fluorescence Excitation-Spectra of Indole in Vapor and Jet - ^1L_a States*, J. Phys. Chem., **94**, 7340, (1990).

[20] J. R. Cable, *Polarization Resolved 2-Photon Resonant Ionization Spectroscopy of Indole and 3-Methylindole*, J. Chem. Phys., **92**, 1627, (1990).

[21] T. L. O. Barstis, L. I. Grace, T. M. Dunn, and D. M. Lubman, *Vibronic Analysis of Indole and 1H-indole-d_6*, J. Phys. Chem., **97**, 5820, (1993).

[22] G. Berden, W. L. Meerts, and E. Jalviste, *Rotationally Resolved Ultraviolet Spectroscopy of Indole, Indazole, and Benzimidazole - Inertial Axis Reorientation in the $S_1(^1L_b) \leftarrow S_0$ Transitions*, J. Chem. Phys., **103**, 9596, (1995).

[23] J. Kupper, D. W. Pratt, W. L. Meerts, C. Brand, J. Tatchen, and M. Schmitt, *Vibronic Coupling in Indole: II. Investigation of the ^1L_a-^1L_b Interaction Using Rotationally Resolved Electronic Spectroscopy*, Phys. Chem. Chem. Phys., **12**, 4980, (2010).

[24] B. J. Fender, D. M. Sammeth, and P. R. Callis, *Site-Selective Photoselection Study of Indole in Argon Matrix - Location of the ^1L_a Origin*, Chem. Phys. Lett., **239**, 31, (1995).

[25] V. A. Povedailo and D. L. Yakovlev, *Electronic Energy of the ^1L_a Level of Supersonic Jet-Cooled Indole*, J. Appl. Spectrosc., **75**, 336, (2008).

[26] C. Brand, J. Kupper, D. W. Pratt, W. L. Meerts, D. Krugler, J. Tatchen, and M. Schmitt, *Vibronic Coupling in Indole: I. Theoretical Description of The ^1L_a-^1L_b Interaction and the Electronic Spectrum*, Phys. Chem. Chem. Phys., **12**, 4968, (2010).

[27] M. N. R. Ashfold, G. A. King, D. Murdock, M. G. D. Nix, T. A. A. Oliver, and A. G. Sage, *πσ* Excited States in Molecular Photochemistry*, Phys. Chem. Chem. Phys., **12**, 1218, (2010).

[28] L. SerranoAndres and B. O. Roos, *Theoretical Study of the Absorption and Emission Spectra of Indole in the Gas Phase and in a Solvent*, J. Am. Chem. Soc., **118**, 185, (1996).

[29] A. L. Sobolewski and W. Domcke, *Ab Initio Investigations on the Photophysics of Indole*, Chem. Phys. Lett., **315**, 293, (1999).

[30] A. L. Sobolewski and W. G. Domcke, *Computational Studies of the Photophysics of Hydrogen-Bonded Molecular Systems*, J. Phys. Chem. A, **111**, 11725, (2007).

[31] A. L. Sobolewski and W. Domcke, *Conical Intersections Induced by Repulsive ^{1}πσ* States in Planar Organic Molecules: Malonaldehyde, Pyrrole and Chlorobenzene as Photochemical Model Systems*, Chem. Phys., **259**, 181, (2000).

[32] B. C. Dian, A. Longarte, and T. S. Zwier, *Hydride Stretch Infrared Spectra in the Excited Electronic States of Indole and its Derivatives: Direct Evidence for the ^{1}πσ* State*, J. Chem. Phys., **118**, 2696, (2003).

[33] M. F. Lin, C. M. Tseng, Y. T. Lee, and C. K. Ni, *Photodissociation Dynamics of Indole in a Molecular Beam*, J. Chem. Phys., **123**, 124303, (2005).

[34] M. G. D. Nix, A. L. Devine, B. Cronin, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy of the near UV Photolysis of Indole: Dissociation Via the ^{1}πσ* State*, Phys. Chem. Chem. Phys., **8**, 2610, (2006).

[35] A. Iqbal and V. G. Stavros, *Exploring the Time Scales of H-Atom Elimination from Photoexcited Indole*, J. Phys. Chem. A, **114**, 68, (2010).

[36] H. Lippert, H. H. Ritze, I. V. Hertel, and W. Radloff, *Femtosecond Time-Resolved Analysis of the Photophysics of the Indole Molecule*, Chem. Phys. Lett., **398**, 526, (2004).

[37] S. T. Park, A. Gahlmann, Y. G. He, J. S. Feenstra, and A. H. Zewail, *Ultrafast Electron Diffraction Reveals Dark Structures of the Biological Chromophore Indole*, Angew. Chem. Int. Edit., **47**, 9496, (2008).

[38] M. A. El-Sayed, *Spin-Orbit Coupling and the Radiationless Processes in Nitrogen Heterocyclics*, J. Chem. Phys., **38**, 2834, (1963).

[39] J. Catalán, P. Perez, and A. U. Acuña, *Indole Spectroscopy - the Location of the ^1L_a and ^1L_b Electronic States and the Absorption-Spectrum*, J. Mol. Struct., **142**, 179, (1986).

[40] D. Robinson, N. A. Besley, E. A. M. Lunt, P. O'Shea, and J. D. Hirst, *Electronic Structure of 5-Hydroxyindole: From Gas Phase to Explicit Solvation*, J. Phys. Chem. B, **113**, 2535, (2009).

[41] S. Arnold, L. Tong, and M. Sulkes, *Fluorescence Lifetimes of Substituted Indoles in Solution and in Free Jets - Evidence for Intramolecular Charge-Transfer Quenching*, J. Phys. Chem., **98**, 2325, (1994).

[42] Y. H. Huang and M. Sulkes, *Anomalously Short Fluorescence Lifetimes in Jet Cooled 4-Hydroxyindole. Evidence for Excited State Tautomerism and Proton Transfer in Clusters*, Chem. Phys. Lett., **254**, 242, (1996).

[43] M. Sulkes and I. Borthwick, *Enhanced Photophysical Effects in Indole Due to C-6 Chemical Group Substitutions*, Chem. Phys. Lett., **279**, 315, (1997).

[44] T. A. A. Oliver, G. A. King, and M. N. R. Ashfold, *Position Matters: Competing O-H and N-H Photodissociation Pathways in Hydroxy- and Methoxy-Substituted Indoles*, Phys. Chem. Chem. Phys., **13**, 14646, (2011).

[45] R. Livingstone, O. Schalk, A. E. Boguslavskiy, G. R. Wu, L. T. Bergendahl, A. Stolow, M. J. Paterson, and D. Townsend, *Following the Excited State Relaxation Dynamics of Indole and 5-Hydroxyindole Using Time-Resolved Photoelectron Spectroscopy*, J. Chem. Phys., **135**, 194307, (2011).

[46] U. Even, J. Jortner, D. Noy, N. Lavie, and C. Cossart-Magos, *Cooling of Large Molecules Below 1 K and He Clusters Formation*, J. Chem. Phys., **112**, 8068, (2000).

[47] P. Kruit and F. H. Read, *Magnetic-Field Parallelizer for 2-π Electronspectrometer and Electron-Image Magnifier*, J. Phys. E. Sci. Instrum., **16**, 313, (1983).

[48] S. Lochbrunner, J. J. Larsen, J. P. Shaffer, M. Schmitt, T. Schultz, J. G. Underwood, and A. Stolow, *Methods and Applications of Femtosecond Time-Resolved Photoelectron Spectroscopy*, J. Elec. Spec., **112**, 183, (2000).

[49] J. F. Stanton and R. J. Bartlett, *The Equation of Motion Coupled-Cluster Method - a Systematic Biorthogonal Approach to Molecular-Excitation Energies, Transition-Probabilities, and Excited-State Properties*, J. Chem. Phys., **98**, 7029, (1993).

[50] O. Christiansen, H. Koch, and P. Jørgensen, *Perturbative Triple Excitation Corrections to Coupled Cluster Singles and Doubles Excitation Energies*, J. Chem. Phys., **105**, 1451, (1996).

[51] *Gaussian 09, Revision A.2, M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, Ö. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, Gaussian, Inc., Wallingford CT, 2009*

[52] *DALTON, a molecular electronic structure program, Release 2.0 (2005), see http://www.kjemi.uio.no/software/dalton/dalton.html 2005.*

[53] N. A. Borisevich and T. F. Raichuyonok, *Electronic Spectra of Indole Vapors*, Dokl. Phys., **52**, 405, (2007).

[54] M. R. Eftink, L. A. Selvidge, P. R. Callis, and A. A. Rehms, *Photophysics of Indole-Derivatives - Experimental Resolution of L_a and L_b Transitions and Comparison with Theory*, J. Phys. Chem., **94**, 3469, (1990).

[55] B. Albinsson and B. Norden, *Excited-State Properties of the Indole Chromophore - Electronic-Transition Moment Directions from Linear*

Dichroism Measurements - Effect of Methyl and Methoxy Substituents, J. Phys. Chem., **96**, 6204, (1992).

[56] J. Hager, M. Ivanco, M. A. Smith, and S. C. Wallace, *Solvation Effects in Jet-Cooled Vanderwaals Clusters - 2-Color Threshold Photoionization Spectroscopy of Indole, Indole Argon, Indole Methane, Indole Water and Indole Methanol*, Chem. Phys. Lett., **113**, 503, (1985).

[57] T. Vondrak, S. Sato, and K. Kimura, *Cation Vibrational Spectra of Indole and Indole-Argon Van Der Waals Complex. A Zero Kinetic Energy Photoelectron Study*, J. Phys. Chem. A, **101**, 2384, (1997).

[58] J. E. Braun, T. L. Grebner, and H. J. Neusser, *Van der Waals versus Hydrogen-Bonding in Complexes of Indole with Argon, Water, and Benzene by Mass-Analyzed Pulsed Field Threshold Ionization*, J. Phys. Chem. A, **102**, 3273, (1998).

[59] M. de Groot, J. Broos, and W. J. Buma, *Tagging Multiphoton Ionization Events by Two-Dimensional Photoelectron Spectroscopy*, J. Chem. Phys., **126**, (2007).

[60] J. H. D. Eland, *Photoelectron Spectra of Conjugated Hydrocarbons and Heteromolecules*, Int. J. Mass Spec. Ion Phys., **2**, 471, (1969).

[61] L. J. Dolby, G. Hanson, and T. Koenig, *He-I Photoelectron-Spectra of N-Methylisoindole and N-Methylindole*, J. Org. Chem., **41**, 3537, (1976).

[62] L. N. Domelsmith, L. L. Munchausen, and K. N. Houk, *Photoelectron-Spectra of Psychotropic-Drugs .1. Phenethylamines, Tryptamines, and LSD* J. Am. Chem. Soc., **99**, 4311, (1977).

[63] M. Kubota and T. Kobayashi, *Electronic Structures of Melatonin and Related Compounds Studied by Photoelectron Spectroscopy*, J. Elec. Spec., **128**, 165, (2003).

[64] O. Schalk, A. E. Boguslavskiy, and A. Stolow, *Substituent Effects on Dynamics at Conical Intersections: Cyclopentadienes*, J. Phys. Chem. A, **114**, 4058, (2010).

[65] H. Satzger, D. Townsend, M. Z. Zgierski, S. Patchkovskii, S. Ullrich, and A. Stolow, *Primary Processes Underlying the Photostability of Isolated DNA Bases: Adenine*, P. Natl. Acad. Sci. USA, **103**, 10196, (2006).

[66] Y. Yamamoto and J. Tanaka, *Polarized Absorption-Spectra of Crystals of Indole and Its Related Compounds*, B. Chem. Soc. Jpn., **45**, 1362, (1972).

[67] E. Jalviste and N. Ohta, *Stark Absorption Spectroscopy of Indole and 3-Methylindole*, J. Chem. Phys., **121**, 4730, (2004).

[68] B. Valeur and G. Weber, *Resolution of Fluorescence Excitation Spectrum of Indole into ^1L_a and ^1L_b Excitation Bands*, Photochem. Photobiol., **25**, 441, (1977).

[69] R. L. Rich, Y. Chen, D. Neven, M. Negrerie, F. Gai, and J. W. Petrich, *Steady-State and Time-Resolved Fluorescence Anisotropy of 7-Azaindole and Its Derivatives*, J. Phys. Chem., **97**, 1781, (1993).

[70] N. Glasser and H. Lami, *Nonradiative Decay of Indoles under Collision-Free Conditions*, J. Chem. Phys., **74**, 6526, (1981).

[71] H. Lippert, H. H. Ritze, I. V. Hertel, and W. Radloff, *Femtosecond Time-Resolved Hydrogen-Atom Elimination from Photoexcited Pyrrole Molecules*, ChemPhysChem, **5**, 1423, (2004).

[72] J. Wei, A. Kuczmann, J. Riedel, F. Renth, and F. Temps, *Photofragment Velocity Map Imaging of H Atom Elimination in the First Excited State of Pyrrole*, Phys. Chem. Chem. Phys., **5**, 315, (2003).

[73] B. Cronin, M. G. D. Nix, R. H. Qadiri, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy Studies of the Near Ultraviolet Photolysis of Pyrrole*, Phys. Chem. Chem. Phys., **6**, 5031, (2004).

[74] A. L. Sobolewski and W. Domcke, *Photophysics of Eumelanin: Ab Initio Studies on the Electronic Spectroscopy and Photochemistry of 5,6-Dihydroxyindole*, ChemPhysChem, **8**, 756, (2007).

3. Experimental setup, Hardware control and Data Processing Software

This chapter describes the newly designed and constructed velocity map imaging photoelectron spectrometer at Heriot Watt University, along with the purpose built data acquisition and analysis software. The lab was empty at the beginning of my PhD and a major part of my project has been to set up the equipment and software required to collect and analyse the experimental data (overviewed in Figure 3.1). This chapter explores the experimental setup, as summarised briefly in this section, in more detail.

The optical setup consists of a titanium sapphire ultrafast pulsed laser, whose principles of operation have been described in Section 1.4. This laser pumps an optical parametric amplifier (OPA), a fourth harmonic box, second and third harmonic setups, and a non-collinear optical parametric amplifier (NOPA). These three lines give three sources of UV light, of which two are used for any one experiment. Of the two UV lines chosen, one is retro-reflected from a computer controlled translation stage. This gives precise control of the length of the path, thus controlling the time delay between the two UV pulses. The two lines, denoted “pump” and “probe” then pass through two shutters. These shutters allow the computer to select which beams to let into the spectrometer at any given time, allowing collection of pump-alone and probe-alone background images, for the purpose of dynamic background subtraction, in addition to the main pump-probe images. Once the beams pass through the shutters they are focussed into the ultra high vacuum interaction region inside the spectrometer. Here they interact with an ultra-cold molecular beam of molecules, as expanded upon in Section 1.3.5. The pump pulse excites the molecules and the probe pulse ionises them. The cloud of ejected electrons or ions is guided towards a microchannel plate which amplifies the signal. It then hits a phosphor screen which converts the signal into light that is collected by a CCD camera. Over many laser shots a picture is built up that provides information about the kinetic energy and angular distribution of the charged particles. If this is repeated for many different pump-probe time delays, a picture of how the charged particle energies and distributions change as the molecule relaxes after photo-excitation can be obtained.

In this chapter, Section 1 describes all the different software components required to collect, process, and analyse the data. Section 2 looks at the optical setup, including the laser system, and the different methods of generating ultraviolet light. The final section looks at the spectrometer itself, including the design of the ion optics. The apparatus and software described in this chapter were used to collect and analyse the results described

in Chapter 4. They are currently being used by another PhD student and will continue to be used to study many different systems after I have left.

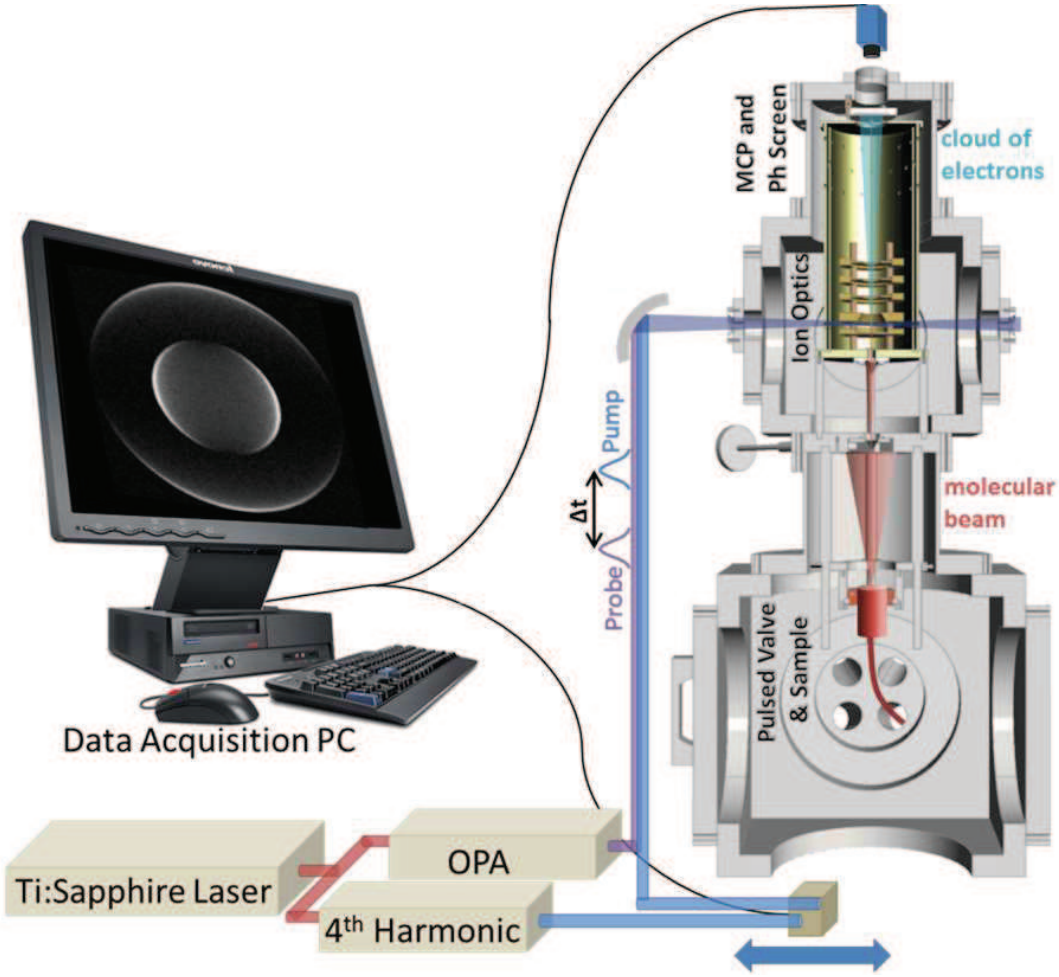


Figure 3.1: Cartoon of the experimental setup.

3.1. Optical System

Time resolved spectroscopy requires very short pulses and the ability to change the wavelength of these pulses at will. This is not a trivial matter and this section describes the infrastructure used to obtain and manipulate these ultrafast pulses. An overview of the setup is outlined here. Details about the laser system are given in Section 3.1.1, the following sections deal with the advantages and problems faced when dealing with ultrafast pulses and non-linear optics, and the final two sections go into more detail on two of the methods of UV light production.

An overview of the optical bench showing the different beam lines in different colours can be found in Figure 3.2. All of these components apart from the NOPA setup

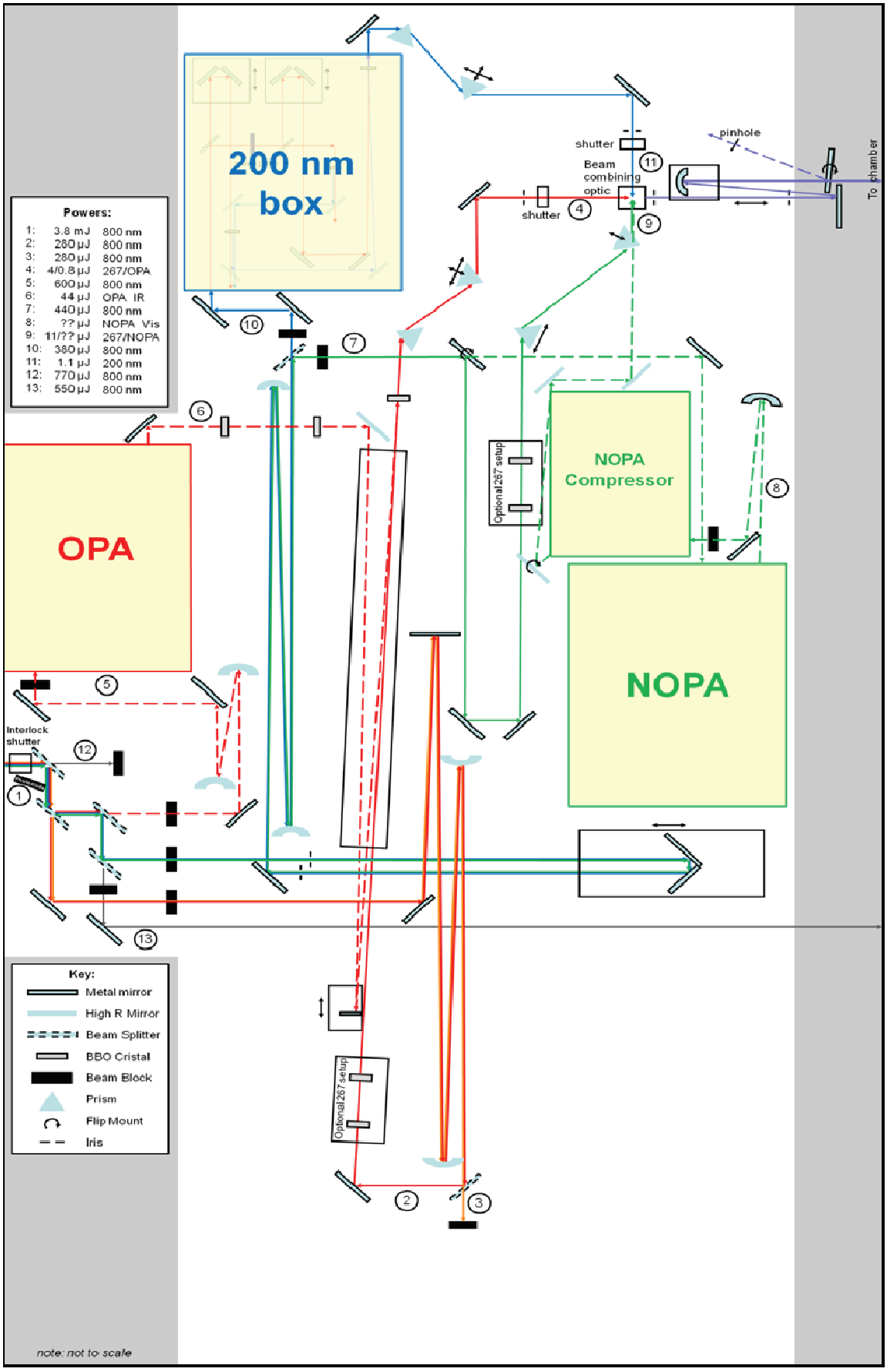


Figure 3.2: Schematic of the optical setup, as it is laid out on the optical bench, with numbered power measuring points.

and the commercial OPA were built by me. To run an experiment, two different beams are required to provide the pump and the probe pulses. However, to increase flexibility and reduce down time between experiments, three main beam lines have been set up to be used in the spectrometer. The 'blue' line goes through the fourth harmonic box (described in 3.2.2) and provides UV pulses of 200 nm wavelength. The 'red' line goes through the optical parametric amplifier (OPA) (described in 3.2.1) to provide tuneable UV pulses. The 'green' line goes through the NOPA and also provides tuneable UV pulses. Both the green line and the red line have the option of bypassing the tuneable setup and creating pulses of 400 nm or 267 nm wavelength instead. The blue and green lines go through a computer controlled translation stage. This is controlled by the data acquisition software and changes the pump-probe time delay. For this reason, for any one experiment either the blue line or the green line must be used in conjunction with the red line.

In order to create the different wavelengths required, the beam diameter coming from the laser has to be reduced. Reducing the beam diameter increases the beam intensity, which increases the efficiency of the non-linear processes employed to create new wavelengths. This allows the beam to travel through small optics such as crystals and prisms without losing power due to the edges of the beam being obstructed. Reducing the beam diameter was done using various telescopes; In order not to lengthen the pulse duration by passing it through dispersive material, concave and convex mirrors were used instead of more traditional lenses. The blue and green lines pass through the same telescope that can be seen running vertically beside the OPA box. All the lines are passed through prism compressors to counteract the effects of dispersion. See Section 3.1.2 for more details of pulse lengthening and compression.

The beams are overlapped on the beam combining optic (labelled (9) on Figure 3.2), which transmits the red line and reflects either the blue or green line depending on orientation. This optic must be chosen carefully for each experiment, with the correct coating that will transmit and reflect the desired wavelengths. Ideally, the shorter wavelength should be reflected, such as for UV light, since shorter wavelengths are more temporally stretched by passing through an optical medium. This can be seen in Figure 3.9 in Section 3.1.5. The beams are then focussed into the interaction region of the Spectrometer through a 2 mm thick CaF_2 window. In our setup we made extensive use of BaB_2O_4, Barium Borate crystals as a non-linear medium. The SNLO software [1, 2] was used to calculate the phase matching angles required for each process.

3.1.1. Commissioning of Femtosecond Laser

The ultrafast pulsed regeneratively amplified Ti:Sapphire laser system was purchased from Spectra Physics, and it's layout is shown in Figure 3.3. It operates with 50 - 100 fs pulse duration, 1 kHz maximum repetition rate, and 750 - 840 nm central wavelength. The laser oscillator [3] is pumped by a 5W Nd:YLF diode laser [4] to give an output of ~ 540 mW. This is then used to seed the regenerative amplifier [5], which is pumped by a 20 W Nd:YLF diode laser [6] to give 4 W total output power. The output of the amplifier is split several ways, as is illustrated by Figure 3.2. One section of the output beam is guided into an optical parametric amplifier [7], also provided by Spectra Physics. More details of this setup can be found in section 3.1.5. After installation, various teething problems required re-alignment and other maintenance to be performed on most of the components of the laser system. The knowledge gained through this process has been summarised in Appendix C, Lab and Laser Procedures. Once these teething problems had been overcome, the laser has worked fairly reliably with good pointing and power stability during experimental runs.

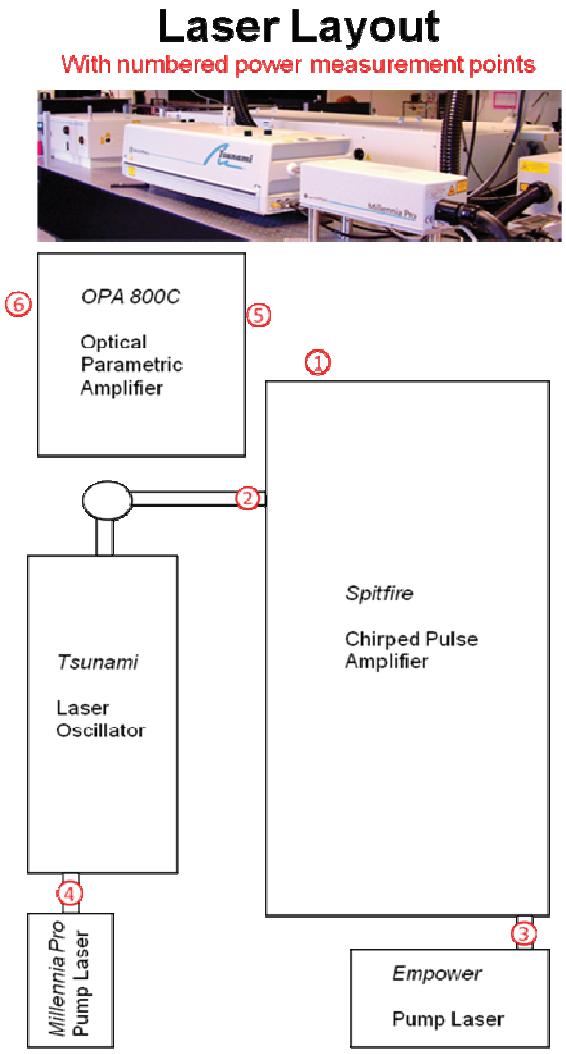


Figure 3.3: Layout of Spectra Physics femtosecond laser system.

3.1.2. Dispersion and Compression of Pulses

One problem when working with ultrafast pulses is pulse lengthening due to optical dispersion, or "chirping" [8-13]. Optical dispersion is due to the fact that the refractive index of all optical materials varies with wavelength. This causes different wavelengths to travel at different speeds and take different paths through optics. When dealing with a continuous wave laser that is close to monochromatic this does not have a large effect, but when an ultrafast pulse with a broad bandwidth passes through any optical material the different wavelengths will become temporally separated or "chirped". This can cause the pulse to become much longer than when it initially left the laser. For example if a 10 fs pulse centred at 200 nm passes through 13 mm of fused silica it will be stretched to 1 ps. This effect is most pronounced with the broad bandwidths required to create femtosecond or attosecond pulses and with UV or IR wavelengths, with some pulses becoming chirped simply by travelling through air. This problem can be overcome by using prisms [8, 14], gratings [14, 15], and chirped mirrors [13, 16]. These methods all apply a "negative chirp", effectively slowing the redder wavelengths down so that all wavelengths are once again temporally overlapped. In the case of the prism compressor setup displayed in Figure 3.4, the blue wavelengths are refracted more, creating a longer path through the second prism material for the red wavelengths. Provided the prism separation (*l*) is optimised correctly, this negative dispersion can be used to cancel out previous positive dispersion, or pre-compensate for future dispersion. Calculating the optimum prism separation involves understanding how the refractive index changes with wavelength, according to Sellmeir's equation [12, 17] (Equation 3.1).

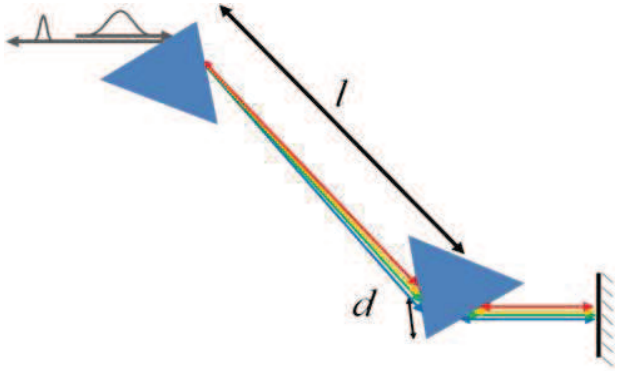


Figure 3.4: Prism compressor.

$$n(\lambda) = \sqrt{1 + \frac{C_1 \lambda^2}{\lambda^2 - C_2} + \frac{C_3 \lambda^2}{\lambda^2 - C_4} + \frac{C_5 \lambda^2}{\lambda^2 - C_6}} \tag{3.1}$$

$$\frac{dn}{d\lambda} = \frac{1}{2n} - \frac{2\, C_1 C_2\, \lambda}{(\lambda^2 - C_2)^2} - \frac{2\, C_3 C_4\, \lambda}{(\lambda^2 - C_4)^2} - \frac{2\, C_5 C_6\, \lambda}{(\lambda^2 - C_6)^2} \tag{3.2}$$

$$\frac{\mathrm{d}^2\mathrm{n}}{\mathrm{d}\lambda^2} = \frac{1}{2\,n^3}\left(\frac{\mathrm{C}_1\,\lambda^3}{(\lambda^2-\mathrm{C}_2)^2}+\frac{\mathrm{C}_1\,\lambda}{\lambda^2-\mathrm{C}_2}-\frac{\mathrm{C}_3\,\lambda^3}{(\lambda^2-\mathrm{C}_4)^2}+\frac{\mathrm{C}_3\,\lambda}{\lambda^2-\mathrm{C}_4}-\frac{\mathrm{C}_5\,\lambda^3}{(\lambda^2-\mathrm{C}_6)^2}+\frac{\mathrm{C}_5\,\lambda}{\lambda^2-\mathrm{C}_6}\right)^2 + \frac{1}{\mathrm{n}}\left(\frac{4\,\mathrm{C}_1\,\lambda^4}{(\lambda^2-\mathrm{C}_2)^3}-\frac{5\,\mathrm{C}_1\,\lambda^2}{(\lambda^2-\mathrm{C}_2)^2}+\frac{\mathrm{C}_1}{\lambda^2-\mathrm{C}_2}+\frac{4\,\mathrm{C}_3\,\lambda^4}{(\lambda^2-\mathrm{C}_4)^3}-\frac{5\,\mathrm{C}_3\,\lambda^2}{(\lambda^2-\mathrm{C}_4)^2} + \frac{\mathrm{C}_3}{\lambda^2-\mathrm{C}_4}+\frac{4\,\mathrm{C}_5\,\lambda^4}{(\lambda^2-\mathrm{C}_6)^3}-\frac{5\,\mathrm{C}_5\,\lambda^2}{(\lambda^2-\mathrm{C}_6)^2}+\frac{\mathrm{C}_5}{\lambda^2-\mathrm{C}_6}\right) \tag{3.3}$$

where C_n are the Sellmeir coefficients for the relevant material, λ is wavelength, and $n(\lambda)$ is the refractive index as a function of wavelength. Once we have these values, many other properties can be calculated, such as the final pulse duration (Δt_f) after passing through an optical medium of thickness p:

$$\Delta t_\mathrm{f} = \Delta t_\mathrm{i}\sqrt{1+\left(\frac{2\ln(2)\,p\,\lambda^3}{\pi\,c^2\,{\Delta t_\mathrm{i}}^2}\,\frac{\mathrm{d}^2\mathrm{n}}{\mathrm{d}\lambda^2}\right)^2} \tag{3.4}$$

the group velocity inside the material (v_g):

$$v_g = \frac{\mathrm{c}}{\mathrm{n}(\lambda)-\lambda\frac{\mathrm{dn}}{\mathrm{d}\lambda}} \tag{3.5}$$

the reflective loss r on a surface for s and p polarised light:

$$r_p = \left(\frac{-\mathrm{n}_1\cos(\theta)+\mathrm{n}_2\sqrt{1-\left(\frac{\mathrm{n}_1}{\mathrm{n}_2}\sin(\theta)\right)^2}}{\mathrm{n}_1\cos(\theta)+\mathrm{n}_2\sqrt{1-\left(\frac{\mathrm{n}_1}{\mathrm{n}_2}\sin(\theta)\right)^2}}\right)^2 \tag{3.6a}$$

$$r_s = \left(\frac{+\mathrm{n}_1\cos(\theta)-\mathrm{n}_2\sqrt{1-\left(\frac{\mathrm{n}_1}{\mathrm{n}_2}\sin(\theta)\right)^2}}{\mathrm{n}_1\cos(\theta)+\mathrm{n}_2\sqrt{1-\left(\frac{\mathrm{n}_1}{\mathrm{n}_2}\sin(\theta)\right)^2}}\right)^2 \tag{3.6b}$$

the group dispersion inside the material ($\mathrm{GDD_p}$):

$$\mathrm{GDD_p} = \frac{1}{4\ln(2)}\sqrt{\left(\frac{\lambda^2\, c_b\, \Delta t_\mathrm{i}}{c\, \Delta\lambda}\right)^2 - \left(\frac{\lambda^2\, c_b}{c\, \Delta\lambda}\right)^4} \tag{3.7}$$

and finally the optimum prism compressor length l:

$$l = \frac{\frac{\mathrm{d}^2\mathrm{n}}{\mathrm{d}\lambda^2}\, d - \frac{6\pi c^2\, \mathrm{GDD_p}}{\lambda^3}}{40\left(\frac{\mathrm{d}^2\mathrm{n}}{\mathrm{d}\lambda^2} + \left(2\mathrm{n} - \frac{1}{\mathrm{n}^3}\right)\left(\frac{\mathrm{dn}}{\mathrm{d}\lambda}\right)^2\right)\sin\left(-2\frac{\mathrm{dn}}{\mathrm{d}\lambda}\Delta\lambda\right) - 80\left(\frac{\mathrm{dn}}{\mathrm{d}\lambda}\right)^2\cos\left(-2\frac{\mathrm{dn}}{\mathrm{d}\lambda}\Delta\lambda\right)} \tag{3.8}$$

Δt_i is the initial pulse duration, $\Delta\lambda$ is the bandwidth of the pulse, θ is the angle of incidence, c_b is the time-bandwidth product (0.44 for a gaussian pulse), d is the distance from the tip of the prism (see Figure 3.4), and c is the speed of light in vacuum.

A program was created to calculate the effects of group velocity dispersion reflective losses, and absorption on ultrafast pulses passing through optical components, and to evaluate all the above equations. The user interface is displayed in Figure 3.5. This assisted in compressing the pump and probe pulses to give a 160 fs cross correlation. The code can be seen in Appendix F.

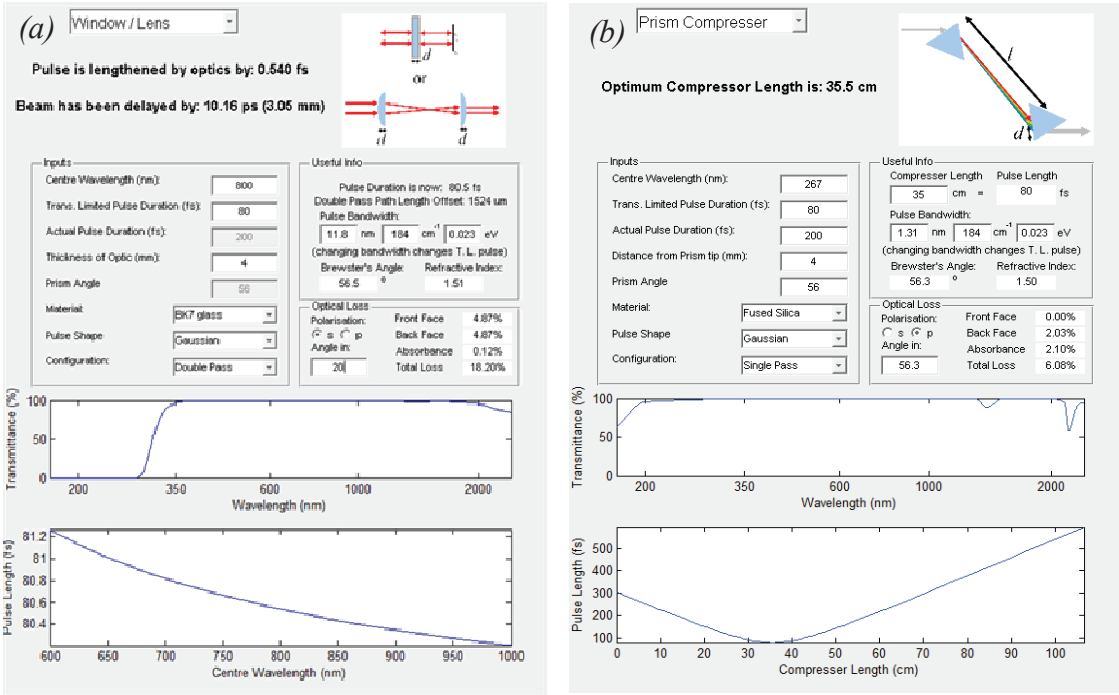


Figure 3.5: User interface for ultrafast optics program. (a) window/lens mode and (b) prism compressor mode.

3.1.3. Fourth Harmonic (200 nm) Setup

In order to produce the fourth harmonic of the Ti:Sapphire output (800 nm) one cannot simply double then double again, as no phase matching angle exists in Barium Borate (BBO) crystals to double 400 nm light. To get around this problem a more complex setup is required (see Figure 3.6). The p-polarised input beam of 800 nm light is split. 80% is passed through a waveplate to change the polarisation and then split again. 50% of this s-polarised beam is used to produce p-polarised 400 nm light by passing through a BBO crystal. The 400 nm light is then mixed with the p-polarised section of 800 nm light on a second BBO crystal. This produces s-polarised 267 nm light, which is subsequently mixed with the final s-polarised 800 nm section to produce p-polarised 200 nm light. Care must be taken to ensure exactly the same path length for each section of light, as for the mixing process to occur the pulses must overlap temporally as well as spatially. In order to assist in this, two delay stages are used to precisely control the path length for each of the mixing processes. As the light passes through the beam splitters and crystals it becomes chirped, lengthening the pulse and reducing the peak intensity. This is problematic because lower peak intensities leads to lower conversion efficiencies for the non-linear processes. In order to avoid this several precautions were taken. The optics were kept as thin as possible: the beam splitters are 3 mm thick; the crystals are 0.5 mm, 0.1 mm and 0.1 mm thick for 400 nm, 267 nm and 200 nm generation respectively; and the wave plate is 0.8 mm thick. These thin optics help reduce the dispersion and walk off within the box.

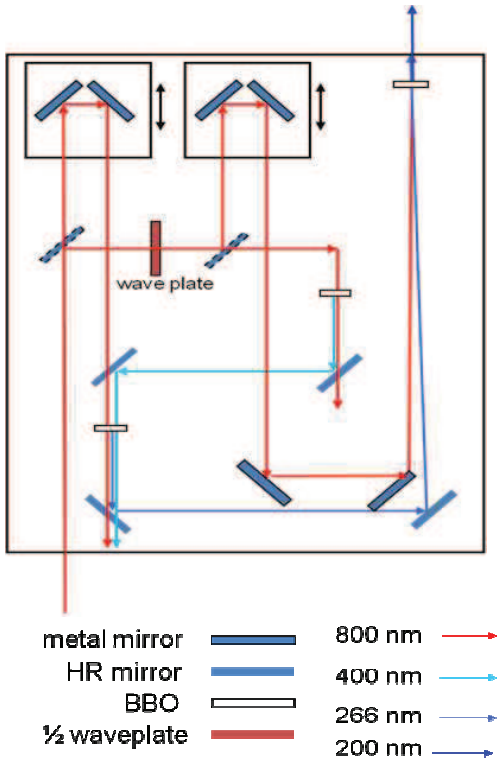


Figure 3.6: 200 box setup.

The final stage was constructed so that the 800 nm light for this process had not passed through any beam splitter, allowing the shorter pulse to effectively ‘gate’ the process. The output from the box is compressed and separated by passing through a prism compressor before entering the chamber. This compensates for any chirp caused by the box as well as pre-compensating for chirp caused by the vacuum chamber window.

3.1.4. 267 nm Setup

The previous section details one way of producing 267 nm light in order to produce 200 nm. This method is effective, however it is difficult to implement and extremely sensitive to the input alignment of the laser beam. This section outlines a simpler method of producing 267 nm light that has also been implemented in the lab (shown as "Optional 267 setup" in two places in Figure 3.2). The method is more stable to beam pointing fluctuations, and is easy to assemble and disassemble, requiring very minimal initial alignment.

As discussed in Section 1.4.2, for type I mixing in a nonlinear crystal the polarisation of the resultant pulse is perpendicular to that of the original pulse(s). When a nonlinear crystal is placed in a beam of p-polarised 800 nm light at the correct angle, a proportion of that beam will be doubled and the output will contain both s-polarised 400 nm and p-polarised 800 nm light. In the setup outlined here, the initial crystal is placed at 45o to the standard angle. It will only double the component of the initial beam polarised along the correct axes. In order to mix the 400 nm with the 800, the 400 nm and the 800 nm should have parallel polarisations (hence the waveplate in the 200 nm setup above). Changing the polarisations to be parallel would require separating the two wavelengths, passing one through a waveplate, then recombining the pulses whilst ensuring the path lengths are exactly the same. This adds unnecessary complexity and beam alignment sensitivity to the setup. However, as demonstrated in Figure 3.7, the polarisation of the two beams is now only 45o apart. A second crystal is placed immediately after the first, and again the components of the initial beams polarised along the correct axes are mixed to form the final 267 nm beam.

Implementing this system adds flexibility to a beam line, as by simply adding or removing crystals the user can use either 800 nm, 400 nm, or the final 267 nm pulse. Changing the wavelength this way has minimal effect on the path length of the beam, which is advantageous in preserving the pump–probe temporal overlap required to run experiments.

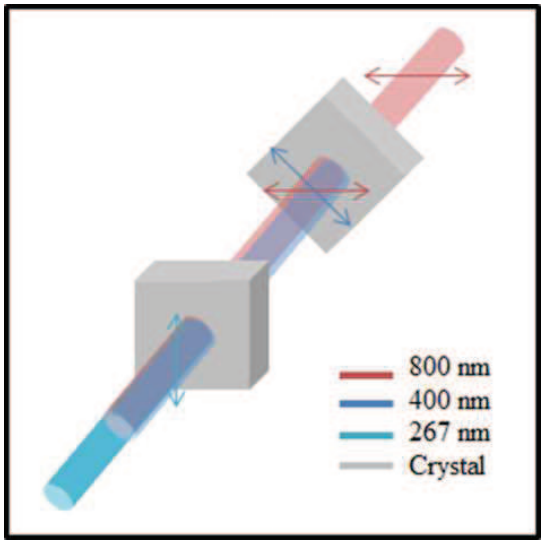


Figure 3.7: Cartoon demonstrating the layout of the 267 nm setup and the relative polarisations angles of the different beams.

3.1.5. Tuneable UV Setup

In order to selectively excite states in a molecule, it is important to have exactly the right wavelength of light. It is also important to be able to easily change between different wavelengths so that time is not wasted between experiments. The following setup was constructed using a commercial Spectra Physics OPA800C optical parametric amplifier (OPA).

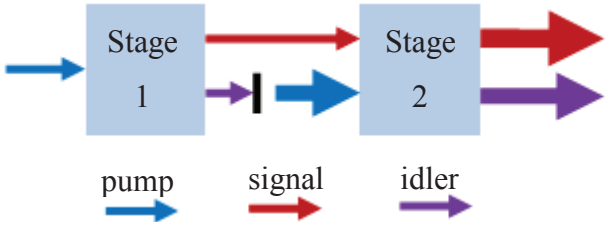


Figure 3.8: Cartoon of the optical parametric amplifier process.

Within the OPA one pump photon is absorbed by a nonlinear crystal, which results in two lower energy photons being emitted. The wavelengths of these two photons can be varied by adjusting the phase matching conditions (i.e. the angle of the crystal). By this process a wide range of infra-red wavelengths can be obtained. The wavelengths of the two resultant output photons (called the signal and idler) are related to the pump wavelength by:

$$\frac{1}{\lambda_{pump}} = \frac{1}{\lambda_{signal}} + \frac{1}{\lambda_{idler}} \tag{3.9}$$

If the signal photons are then passed through a nonlinear crystal again, this time collinearly with photons of the pump beam, each pump photon converts to two photons, one signal and one idler. This process is demonstrated in Figure 3.8. The signal and idler have polarisations that are perpendicular to one another, thus allowing for easy separation of the two beams. If the OPA is pumped with the 800 nm Ti:Sapphire output, then the signal wavelength can range from 1100 nm to 1600 nm, and the idler wavelength can range from 1600 nm to 2900 nm (see Table 3.1).

The output wavelengths of the OPA are much too long for our purposes. The process of converting from the infra-red to the ultra-violet is demonstrated in Figure 3.10. The input to the OPA must be telescoped to reduce the beam size. The output power of the OPA is very sensitive to the pointing of the input beam, which can be adjusted using the two mirrors after telescope 1. The output of the OPA is then passed through two removable BBO crystals. This gives the option of doubling the photon energy (i.e. halving the wavelength) either once, twice, or not at all. Once this is done the pulse travels through an optional collimating telescope, required to correct the divergent OPA output. A divergent beam can be a problem for frequency mixing at the final BBO crystal as the intensity per square metre is much reduced. Care must be taken, however, to avoid

stretching the pulse duration. As can be seen in Figure 3.9, the pulse duration is strongly dependant on wavelength, therefore the collimating telescope should only be used with wavelengths in the visible and near infrared.

The other beam involved in the tuneable UV setup is the mixing beam. This is shown in solid red in Figure 3.10. This is initially sent through telescope two to reduce the beam size. The folded nature of telescope two is required to ensure the path length of this beam is the same as the path length of the beam passing through the OPA. Fine adjustments of this path difference can be made on the translation stage. The path lengths must be exactly the same to within ~0.03 mm in order that the ~100 fs pulses overlap temporally in the crystal and create a frequency mixing effect. The mixing beam can be converted to 400 nm or 267 nm using the optional BBO crystals after the telescope. It is then combined with the OPA beam in the final BBO crystal. All the BBO crystal must have the correct rotation and orientation for phase matching of the particular wavelengths required, and any change on the OPA output wavelength will require small changes in the crystal angles and in the translation stage position.

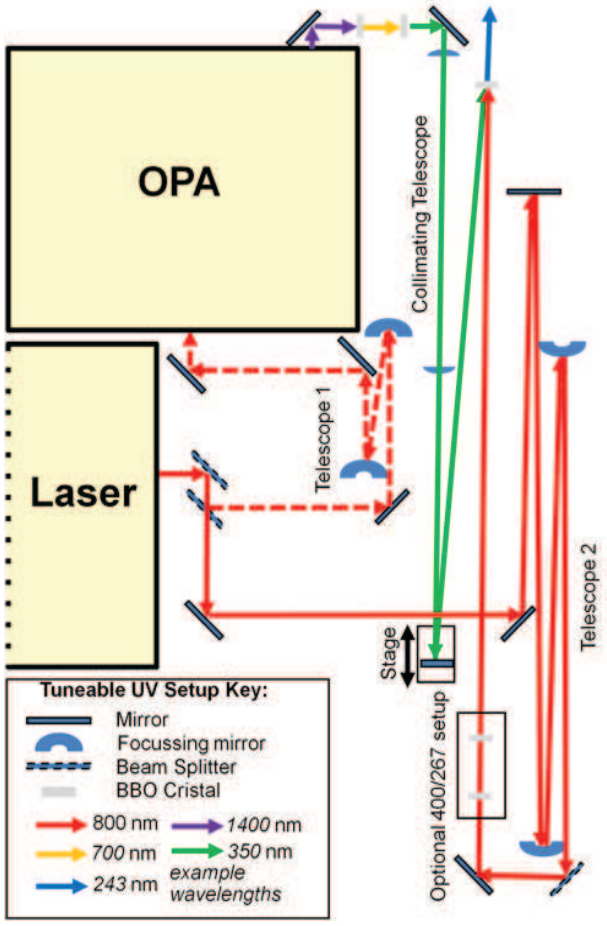


Figure 3.10: Tuneable UV setup.

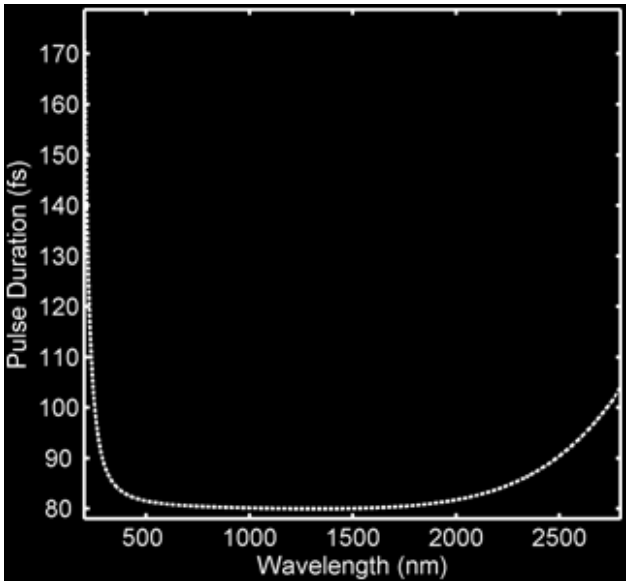
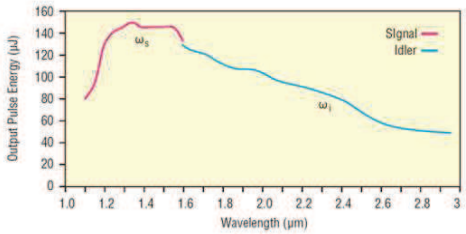


Figure 3.9: Pulse stretching of an 80 fs pulse due to propagation through two 4mm thick glass lenses. Calculated using Equation 3.3.

The tuneable UV setup has been designed to be as flexible as possible. Table 3.1 illustrates a few of the different possible configurations with the range of wavelengths that can be acquired by each configuration. Many wavelengths can be accessed by more than one configuration. When choosing which configuration to use, several factors must be considered. Firstly, the range of wavelengths that will be required for the current experiment, and possibly the next few experiments. If you can find one configuration that covers all the wavelengths needed, this is advantageous as it will require less turn around time between experiments. Secondly, the required pulse intensity. If you are using the tuneable UV as your probe, or if the molecule's vapour pressure or ionisation cross section is low, you may want a more intense pulse. In general, the mixing configurations give more power then those based simply on doubling, but can be less stable to laser pointing fluctuations. When looking at the OPA signal and idler powers on the graph it is important to remember that all that the configurations required to change the wavelength are non-linear processes, so a small drop in output power from the OPA can result in a large drop in power in the final pulse. It is also useful to note that mixing processes are likely to be more successful if the wavelengths are similar to each other due to walk-off considerations (see Section 1.4.3)

To get UV wavelengths
use the **BOLD wavelengths** as indicated,
and dump the GRAY wavelengths



Double twice & mix with 800 nm

UV	Signal	Idler
380	1105	**2895**
370	1128	**2753**
360	1152	**2618**
350	1179	**2489**
340	1209	**2365**
330	1242	**2247**
320	1280	**2133**
310	1323	**2024**
300	1371	**1920**
290	1428	**1820**
280	1493	**1723**
270	1571	**1630**
260	**1541**	1664
250	**1455**	1778
240	**1371**	1920
230	**1291**	2103
220	**1214**	2347
210	**1139**	2688
200	**1067**	3200

Mix with 400 nm

UV	Signal	Idler
345	1174	**2509**
340	1236	**2267**
335	1307	**2062**
330	1389	**1886**
325	1486	**1733**
315	**1482**	1738
310	**1378**	1908
305	**1284**	2122
300	**1200**	2400
295	**1124**	2776

Mix with 267 nm

UV	Signal	Idler
240	1207	**2373**
235	1351	**1961**
230	1544	**1660**
225	**1430**	1815
220	**1250**	2223
215	**1104**	2906

Double then mix with 400 nm

UV	Signal	Idler
315	1096	**2965**
310	1127	**2756**
305	1162	**2568**
300	1200	**2400**
295	1242	**2248**
290	1289	**2109**
285	1341	**1983**
280	1400	**1867**
275	1467	**1760**
270	1543	**1662**
265	**1570**	1631
260	**1486**	1733
255	**1407**	1855
250	**1333**	2000
245	**1265**	2178
240	**1200**	2400
235	**1139**	2686
230	**1082**	3067
225	**1029**	3600

Double then double again

UV	Signal	Idler
460	1415	**1840**
450	1440	**1800**
440	1467	**1760**
430	1496	**1720**
420	1527	**1680**
410	1562	**1640**
390	**1560**	1642
380	**1520**	1689
370	**1480**	1741
360	**1440**	1800
350	**1400**	1867
340	**1360**	1943
330	**1320**	2031
320	**1280**	2133
310	**1240**	2255
300	**1200**	2400
290	**1160**	2578
280	**1120**	2800
270	**1080**	3086

Table 3.1: possible configurations and wavelengths obtainable from the tuneable UV setup (note: only wavelengths between 200 nm and 460 nm are shown, but wavelengths outside this range are also possible).

3.2. Vacuum System

3.2.1. Spectrometer Design

The velocity map imaging photoelectron and photoion spectrometer used in the experiments was designed and built over the course of my PhD. An overview of the system is shown in Figure 3.11. This differentially pumped, ultra-high vacuum spectrometer consists of two main chambers. At the bottom there is the source chamber, with the main interaction chamber mounted on top. The two chambers are connected by a 1 mm skimmer (Beam Dynamics Inc) which is mounted on a translatable gate-valve assembly [18]. Each chamber is evacuated using separate turbomolecular pumps (Edwards Vacuum, 2200 l/s and 480 l/s respectively). The gate-valve allows the source chamber to be vented to atmospheric pressure while the main chamber remains at base pressure (~1×10^{-8} mbar). The main advantage of this setup is that it allows the samples in the source chamber to be replenished or changed without the need for a time consuming bake out of the main chamber to return it to ultra-high vacuum before re-starting experiments. This significantly enhances data collection rates, as a typical turn around time for replacing the sample, from finishing one experiment to recommencing photoelectron acquisition on a new molecule, is only around two hours.

Solid samples are held in a cartridge mounted inside an Evan Lavie [19] pulsed valve inside the source chamber. The valve is a high intensity supersonic pulsed molecular beam valve with a 200 μm diameter conical nozzle. A carrier gas (usually helium at 3 bar pressure) is passed over the heated sample to bring it into the gas phase, and this gas is then expanded in a molecular beam. Normal operation of the solenoid valve driver at 1 kHz raises the temperature of the sample in the valve sufficiently to obtain good levels of photoelectron signal. To regulate the temperature a copper cooling block was placed over the valve body and connected to a closed loop chiller unit (Neslab RTE-110) circulating a 50:50 mixture of ethylene glycol and water. Even with this system in place, it is hard to operate the valve at temperatures below ~50^{o}C and samples with low melting points or liquids cannot be used in this configuration.

In this case there are two options. The repetition rate of the experiment can be reduced. This reduces the heating of the valve but slows down data acquisition times. For the other option the sample holder can be left empty, and the helium gas passed through the sample at room temperature in an external pick up cell. This option is good for molecules with high vapour pressures.

The supersonic expansion from the valve translationally and internally cools the molecules, which then pass through the skimmer into the main chamber. The skimmer, which is radiatively heated using a small halogen lamp to prevent solid deposition on the tip, selects only the coolest molecules to pass into the interaction region. Once in the interaction region, the molecular beam is intersected at 90^{o} by the co-propagating pump and probe laser pulses and the resultant photoelectrons are guided towards the detector by the ion optics setup.

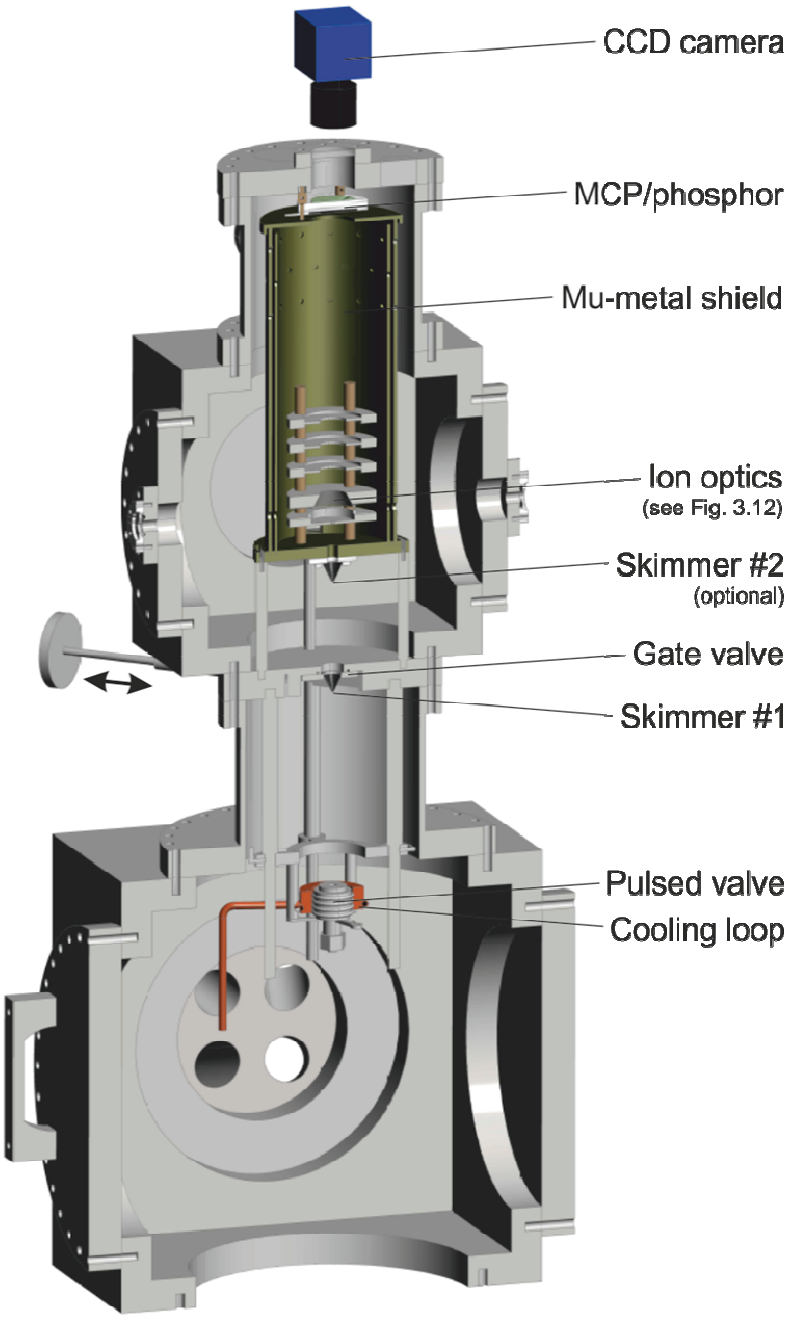


Figure 3.11: Cut-through section of the photoelectron imaging spectrometer used in the current experiments. For additional information see the main text. For clarity, some additional details such as vacuum pumps, bake-out lamps and vacuum feedthroughs are omitted.

3.2.2. Velocity Map Imaging Setup

The ion optics are designed to extract the photoelectrons or photoions from the interaction region and image them onto the detector. As discussed in Chapter 1, velocity map imaging (VMI) [20] allows all electrons emitted with the same direction and energy from a finite volume to be imaged to a single position on the detector. This allows for much greater energy resolution, as the detected position becomes a function of only kinetic energy and angular direction, not dependent on original position. The history and principles of imaging in this way are described in Section 1.3.3. In order to create the best conditions for velocity mapping, the setup described below and shown in Figure 3.12 is employed. I was assisted in this work by Maria Iljina and Ross Donaldson.

The interaction between the UV light pulses and the molecular beam containing the sample takes place between the repeller and extractor electrodes of the electrostatic lens set-up. The design of the lens assembly was modelled using the SIMION software package (Scientific Instrument Services, Version 7.0). A key feature of this arrangement is the use of a conical extractor electrode. Compared to flat, parallel electrode geometries this gives improved velocity mapping of charged particles ejected perpendicular to the time of flight axis, effectively reducing some lens aberration effects. Another key feature is the extruded "lip" on both the extractor and repeller (see Figure 3.12). This creates a potential barrier close to the edge of these lens elements that effectively screens the insulating components between them from charged particles formed in the interaction region. This considerably reduces the possibility of charge build up on these components, which could lead to distorted and non-round images caused by field distortions within the ion optic setup. Similar design features have also previously been employed by other groups [21-23]. The thick outer

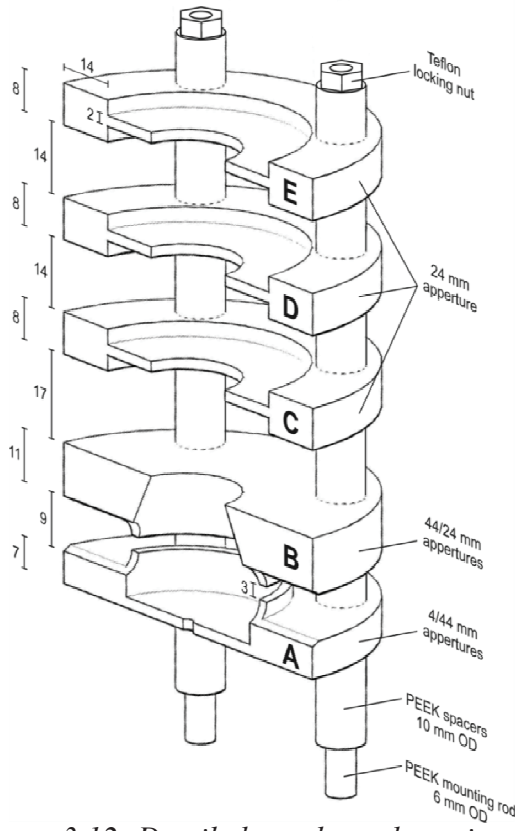


Figure 3.12: Detailed cut-through section of the photoelectron or photoion imaging electrodes. Units are all shown in mm.

edge design of all electrodes was to enable mounting of electrical contacts without potentially distorting the field lines. The additional electrodes (D-E) offer the capability if performing "DC slice imaging" experiments [24], as mentioned in Chapter 1, when running in ion detection mode. When DC slice imaging is not required or possible, these electrodes are grounded along with electrode C to create a field free area between electrode C and the detector. This creates a versatile system that can be used in different modes to extract different kinds of information.

The velocity map imaging electrode assembly and photoelectron flight tube are fully insulated from the effects of stray magnetic fields by a double layer of mu-metal shielding along the time-of flight axis (see Figure 3.11). These shielding cylinders have internal diameters of 102 mm and 112 mm and are 2 mm and 3 mm thick, respectively. There is a mu-metal end-cap and base plate which are 3 mm and 5 mm thick, respectively. The end cap has a 40 mm central hole to allow access to the detector. A series of small (5 mm) holes around the top end of the cylinders ensure adequate gas pumping in the region close to the MCP detector. In order to minimize the effects of stray electric fields within the instrument, all shielding surfaces that could possibly be exposed to the charged particles were spayed with colloidal graphite (Acheson Colloids, Aerodag G). The entire main chamber can also be baked to a temperature in excess of 150^{o}C using small, internally mounted halogen lamps positioned outside the mu-metal shield.

The detector assembly used to detect the photoelectrons and photoions is a 40 mm dual microchannel plate (MCP) that amplifies the signal, backed by a P47 phosphor screen (Photonis, APD 2PS 40/12/10/12 I 46:1 P47) which converts the electrons into light that can be detected by a monochrome CCD camera (The Imaging Source, DMK 21BF04). The MCP and phosphor screen are mounted onto a custom made flange with a central CF40 viewport and three CF16 Safe High Voltage (SHV) feedthroughs, connected to the two MCP plates and the phosphor screen. The voltage on the back plate of the MCP assembly is gated using a high voltage pulser (DEI PVX-4140) to reduce background noise. The timing of the detector (and the pulsed valve) are controlled by a multi-channel delay generator (Stanford, DG535), which is triggered by the software controlling the laser system. This ensures that the pulsed molecular beam is temporally overlapped with the UV pump and probe pulses, and that the detector is turned on when the photo-electrons or photo-ions impact it. The detector is fitted with an interlock that automatically cuts off any voltage to the detector if the chamber pressure reaches a certain level. This is to avoid accidental damage to the sensitive detector.

3.2.3. Initial Commissioning and Calibration

Once the spectrometer and optical setup were fully assembled, various tests and readings had to be taken before data could be acquired and analysed.

In order to observe photoelectron and photoion signal, the temporal overlap of the pump and probe pulses, the molecular beam and the detector must be optimised. As overviewed in Figure 3.13, the timing delay box takes a trigger from the laser and controls the timing of the valve firing (τ_{valve}), the detector turning on (τ_{MCP}) and the detector duration. The optimum timing of τ_{valve} depends on the flight time of the molecular beam, in turn a function of the molecular weight of the gas. Once the valve fires, the molecular beam takes some time to travel from the valve to the interaction

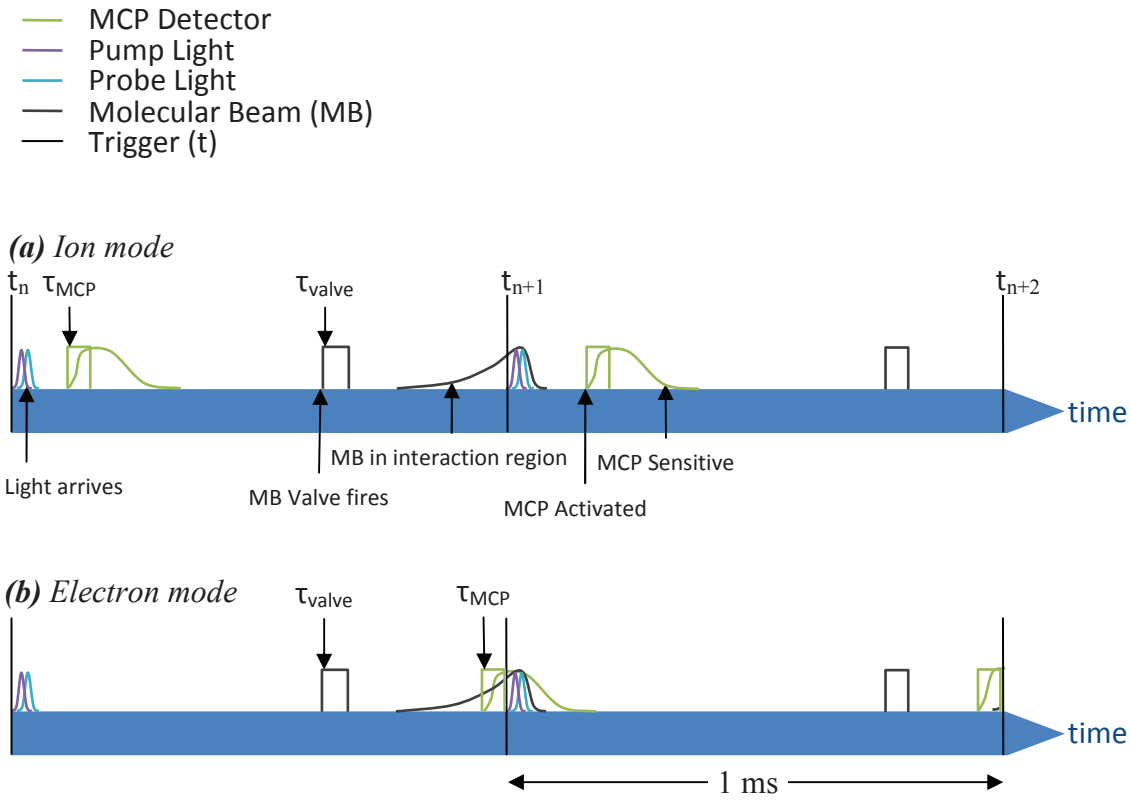


Figure 3.13: Cartoon of temporal profile of experimental setup (not to scale), showing how the temporal overlap of the detector, light, and molecular beam are achieved. τ_{valve} and τ_{MCP} are measured from t_n trigger pulse. (a) shows the setup in ion mode, where the detector must be sensitive some time after the ions are ejected, to account for the finite flight time of the (heavy) ions from interaction region to detector. (b) shows the setup in electron mode, where the detector must be sensitive at the same time as the electrons are ejected.

region. In order to allow for this travel time, the valve is fired after the light pulse, to allow the molecular beam to overlap with the following light pulse. In ion mode, τ_{MCP} occurs some time after the light arrives. By scanning τ_{MCP} we can select for different ion masses (i.e. time-of-flight), thus allowing for the selective detection of only the ion of interest. In electron mode the electrons travel much faster then the ions, arriving almost instantaneously at the detector. It was found that setting τ_{MCP}= 0 did not work as the MCP plates take some time to turn on (as is demonstrated by the curved green line in Figure 3.13), and the detector was missing the electron signal. The signal to the detector (square green line) could not span the following trigger signal (t_{n+1}) as this was not supported by the timing delay generator, so the trigger from the laser was altered to arrive slightly later in time, and τ_{MCP} was set to fire immediately before t_{n+1} as demonstrated in Figure 3.13b. This allowed electron signal to be observed. A typical photoelectron experiment, using helium as the seed gas, has τ_{MCP} of ~997 μs and τ_{valve} of ~720 μs.

In addition to temporal overlap, the molecular beam and the laser beams must be spatially overlapped. The equipment was designed such that aligning the beams through the centre of the entrance and exit windows should bring the laser beams into perpendicular intersection with the molecular beam. Adjusting the laser beam pointing horizontally whilst observing the photo-ion signal allowed optimisation of this overlap. Adjusting the laser beam vertically does not affect the overlap with the molecular beam, but can reduce background signal in electron mode due to photon scattering within the chamber. Vertical adjustment of the laser beam also affects the velocity mapping conditions, so any adjustments must result in a new calibration image being collected before any experiments can be run. Once the optimum horizontal and vertical position was identified, two irises were set in place as an alignment guide and all experiments were conducted using this alignment. The pump and probe beams must both have

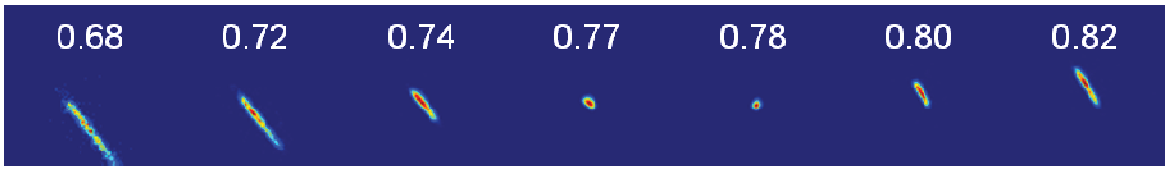


Figure 3.14: Effect of adjusting the VMI voltage ratio on photo-ion signal, showing the image stretching effect of applying the wrong ratio (white text is the ratio applying to the corresponding image).

horizontal polarisation in order for the required cylindrical symmetry to be produced. The polarisation was checked using a polariser and altered using a wave plate to ensure the best possible polarisation.

The quality of the velocity mapping created by the ion optics shown in Figure 3.12 is dependent on the voltage ratio between A and B [20]. The effect of varying this ratio in photo-ion mode is shown in Figure 3.14. It was found for our setup that there was a slightly different ratio for ions (0.7785) than for electrons (0.769). This ratio is important to create a focussed image as demonstrated in Figure 1.14 in Chapter 1. Once the ratio was determined, calibration images were obtained. Increasing the magnitude of the voltages reduces the size of the image. In order to obtain the highest possible resolution for each experiment, the best voltages for A and B must be chosen carefully such that the image is as large as possible without losing information off the edge of the detector. For this reason calibration images were obtained for many different voltage ratios, from A = 500 V up to A = 2500 V. To create a calibration image, a molecule with a well-known photoelectron spectrum is loaded into the spectrometer, and a single image is obtained. This image is then loaded into the data processing and analysis software, where the peaks are identified and assigned to the known energy via the user interface “create a calibration file” option. This then creates a calibration file that can be used for converting between radius and energy (see Section 3.3.3 for more detail on calibration file usage).

3.3. Software Design

In order acquire the data by controlling the hardware, as well as to process and analyse it, various pieces of software had to be designed. This was a major part of my PhD project, taking several years to complete; the final code can be seen in the appendices. It was decided that it would be sensible to use one language throughout, to aid both ease of transfer of data between programs, and ease of programming. MATLAB was chosen for two reasons; firstly it is excellent at handling large sets of data, and secondly because of existing code written by Albert Stolow's group in Ottawa that was a good starting point for one section of the data processing code. The purpose of the code is to provide a single platform that spans the entire process from collecting preliminary data sets to generating the figures for a paper. For more details of the workings of the code and how the user interacts with it, see the user manual in Appendix C. For the code itself, see Appendix B.

All the code written can be categorised into two main sections: (i) hardware control, and (ii) data processing. The former controls the translation stage, shutters and camera required to run the experiment; and acquires the data. The latter (as summarised in Figure 3.15) takes the data, dynamically subtracts the background images, performs the required mathematical transforms, displays the data as a 3D intensity *vs* electron energy *vs* time delay graph, and fits exponential decays and anisotropy parameters. The following sections provide more insight into these programs.

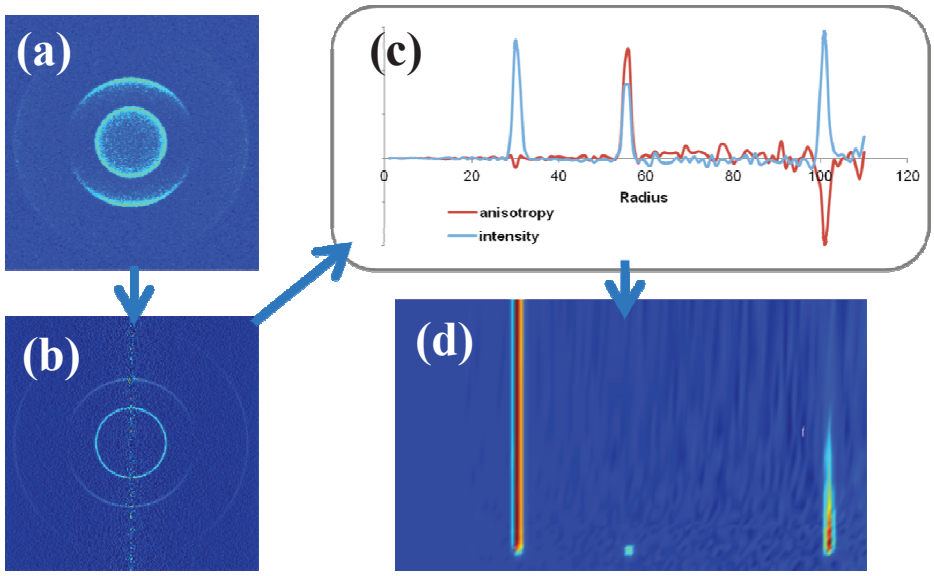


Figure 3.15: Overview of data processing: (a) raw image built up from many images added together and background subtracted; (b) Abel inversion of the image retrieves the central slice; (c) intensity and anisotropy extracted as a function of radius; (d) many images collected as a function of time delay, which when processed builds up a 3D distribution. This distribution can then be fitted to extract decay times. The data shown here is model data generated to test the software.

3.3.1. Hardware Control

The hardware control program panel shown in Figure 3.16 and Figure 3.17 is designed to interface with the translation stage (used to control the time delay between pump and probe), the shutters (used to let the pump beam and the probe beam into the chamber) and the camera (used to acquire the data).

The user interface must be flexible enough to allow the user to do a wide range of tasks. There is the ability to view a live feed from the camera, with a graph of average intensity over time (Figure 3.16). This is useful for the initial setup of the beam lines, making sure the two pulses and the molecular beam are spatially overlapped, and for monitoring the MCP detector as it is slowly turned on, making sure no hot-spots or arcing occurs. While this mode is running, the program continuously adds the images to make an integrated image that is also displayed on the user interface. This is useful for collecting a single image over a long time frame, for example to test velocity mapping conditions or to obtain an electron image when signal levels are low. Both the shutters and the stage can be manipulated manually through the program user interface. In

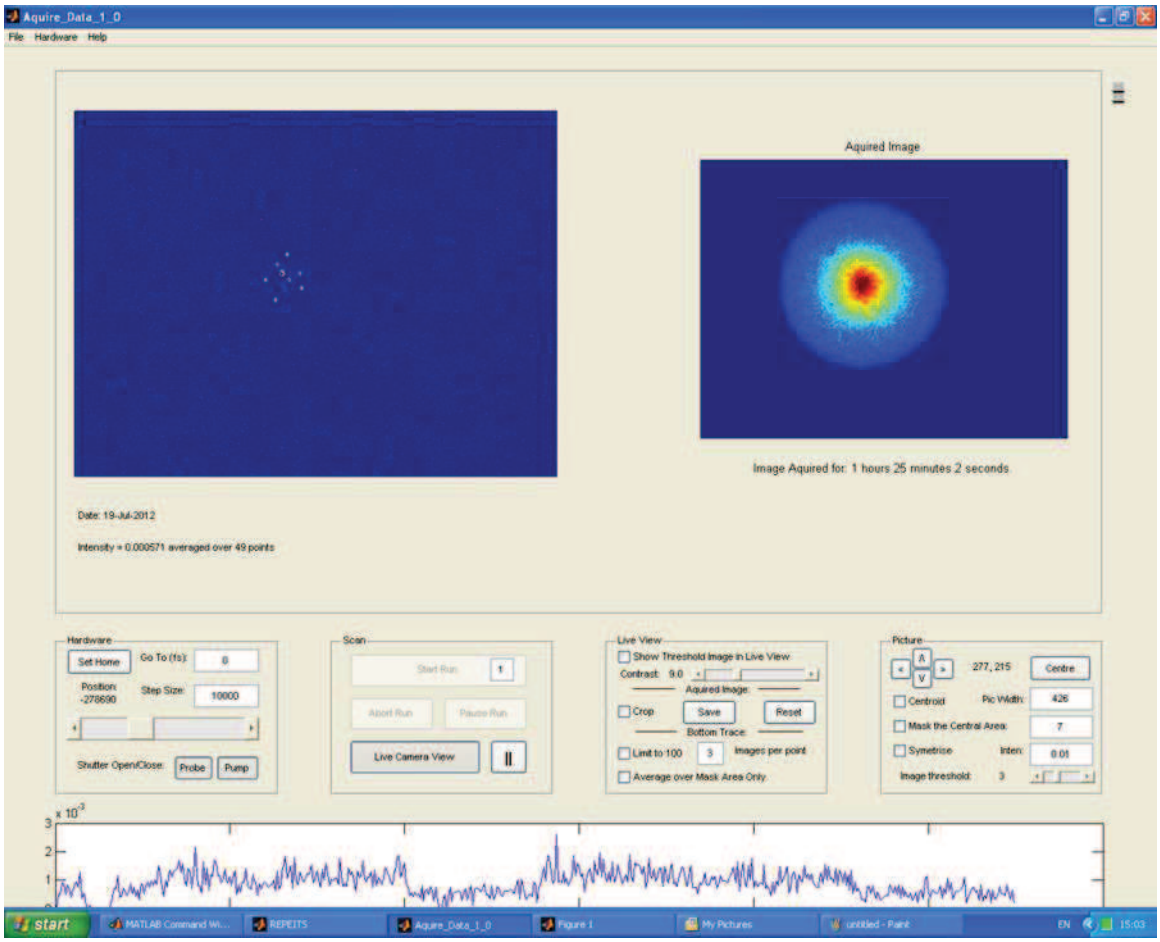


Figure 3.16: Aquire_Data User Interface shown in live view mode whilst collecting a photoelectron image.

conjunction with the live feed from the camera, this can be used to search for the point where the pump and probe beam overlap completely in time (t_0). Once this stage position has been found, the program will allow the user to make this the new zero position on the stage, so that every time delay is measured from this point.

Once everything has been satisfactorily set up, the user can set up a data run consisting of multiple scans through t_0, and to longer time delays. In order to be efficient, the scan can be set up to take linear steps through the region around t_0, then exponentially increasing step sizes for longer time delays. This allows rapid processes to be seen in detail, and processes occurring on a longer time scale to be simultaneously observed without excessive extra steps. At each step, an image is integrated over many laser shots, with an acquire time specified by the user. The software takes all the images from the video stream collected during the time period, applies some preliminary processing detailed below in Equations 3.10-3.12 and surrounding text, and adds them together to create one integrated image. In addition to this pump-probe image, background images can optionally be acquired in the same manner. These images capture the pump-alone and

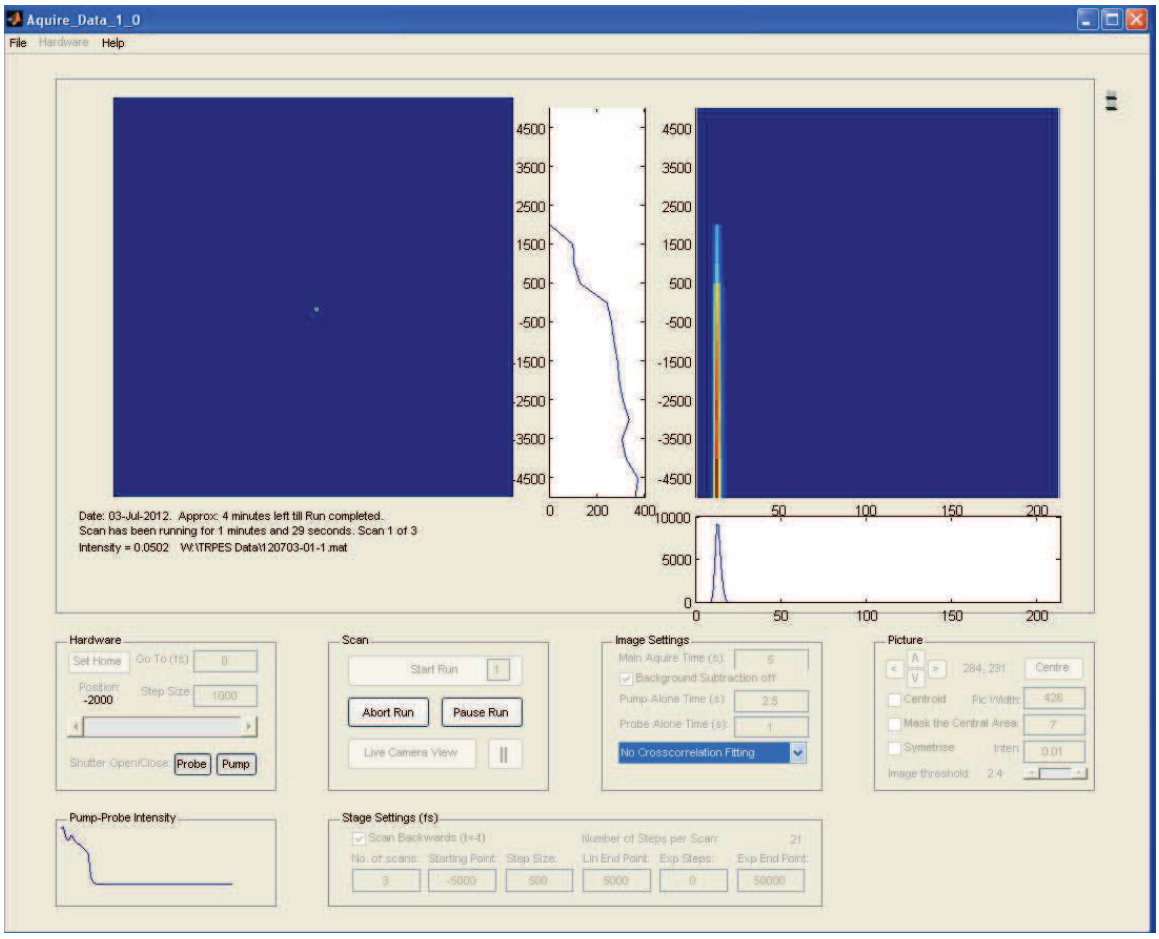


Figure 3.17: Aquire_Data User Interface shown during a data run collecting photoelectron data.

probe-alone signals at each time step. Ideally the pump-alone and probe-alone signals would be invariant with time delay, but this method compensates for laser fluctuations, drift, and changing conditions. These background images are subsequently subtracted from the pump-probe images in the data processing stage to remove unwanted signals. If the laser drifts too much, or there is a problem, the run can be paused at the end of the current scan, the equipment can be modified, and the run set into motion again. At the end of each scan, all the images from that scan are saved in a MATLAB data file. This ensures that if a run has to be aborted or the computer crashes, all the completed scan data is not lost.

While the run is progressing, the user interface displays some of the data so the user can see how the experiment is progressing. It displays the most current pump-probe image and a colour-mapped time delay *vs.* radius *vs.* intensity graph so the user can see what the final data set will look like. To complement the time delay *vs.* radius *vs.* intensity graph, it displays intensity *vs.* time delay and intensity *vs.* radius graphs. The data to be displayed on these side graphs can be selected using the curser on the main colour-mapped graph. The user interface also displays three average intensity *vs.* running time graphs for the three types of image (pump-probe, pump alone and probe alone). This allows the user to monitor if there is a problem with any of the equipment, or if the sample of molecules is running out. Finally, it displays the approximate time left to complete the run. There is an option to fit a Gaussian function to the average intensity *vs.* time delay graph during a scan. The equation used is

$$y = A\, e^{-\left(\frac{2.35\left(x-x_{ofs}\right)}{\sqrt{2}\,FWHM}\right)^2} + y_{ofs} \tag{3.13}$$

This is useful for ascertaining the cross correlation of the pulses. This is important as often the pulses can become stretched out due to frequency dispersion (chirping). Prism compressors are added to the optical setup to counteract this effect, but these must be optimised to ensure the shortest possible pulse duration. The ability to measure the pulse duration is essential for optimising the prism compressors. See Section 3.1.2 Dispersion and Compression of Pulses for more details.

As the images are being acquired, some real-time data processing is desirable, as not all processing can be applied after the images are integrated. When an electron or ion hits the detector it produces a spot of light that is typically about three to five pixels wide. In addition, some spots are brighter than others due to natural fluctuations and inhomogeneity within the detector. In order to counter this, centroiding or ion-counting

methods can be implemented that converts each spot of light into a single pixel of unit intensity. The method chosen is described in [25]. This method takes advantage of the fact that at the centre (maximum intensity) position of each spot, the pixels at each side will have a lower intensity. Occasionally the maximum value of a spot will be spread over two pixels. In this case, only one of those pixels must be chosen. In order for a pixel of intensity $p(x, y)$ to be chosen it needs to satisfy the following inequalities:

$$p(x,y) > threshhold \tag{3.10}$$

$$p(x-1,y) < p(x,y) \geq p(x+1,y) \tag{3.11}$$

$$p(x,y-1) < p(x,y) \geq p(x,y+1) \tag{3.12}$$

The *threshold* is a value chosen to eliminate any low level camera noise, but must not be so high that it eliminates real signal. The threshold value can be set using a slider on the user interface, and is used to improve the image signal-to-noise even when the centroiding option has been turned off. Any pixel that satisfies all the inequalities is given the new value of one, and all other pixels are given the value zero. If more than one spot occurs in the same area such that they overlap, they are often counted as one spot using this method. In order to avoid this problem, each individual image that comes from the camera must be processed before being added together, rather than acquiring the image for some time, then processing. Using MATLAB's matrix operations, this calculation can be performed extremely rapidly, and thus can be applied in real time to a 60 frames per second video stream. An example of images collected with and without this processing is shown in Figure 3.18.

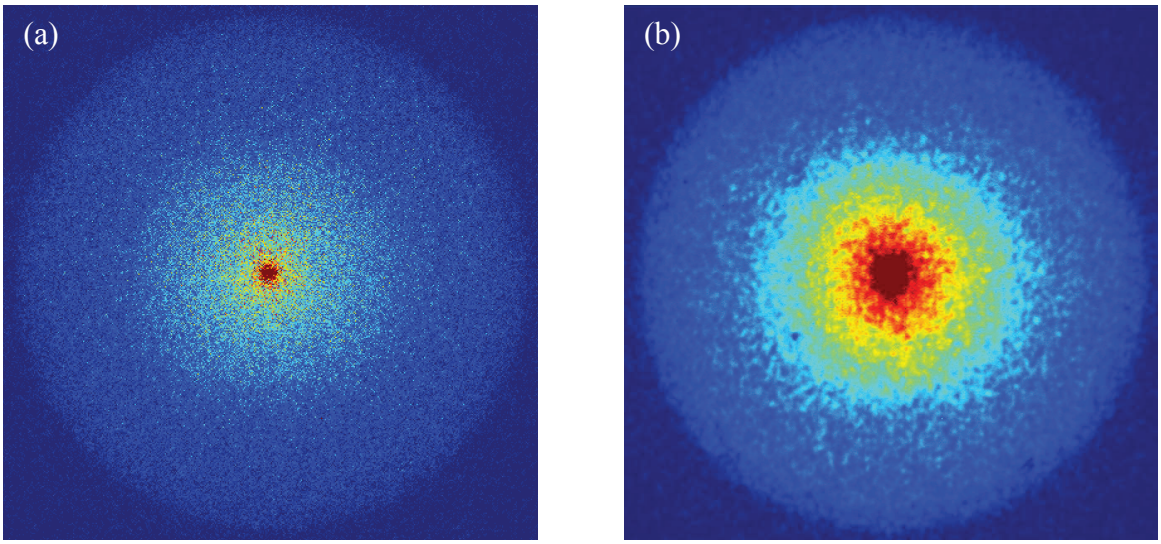


Figure 3.18: Two 1-naphthylamine photoelectron images comparing the effects of centroiding. (a) was taken with centroiding turned on. (b) was taken with centroiding turned off. Both were acquired for 35 minutes.

3.3.2. Initial Data Processing (Process_Data)

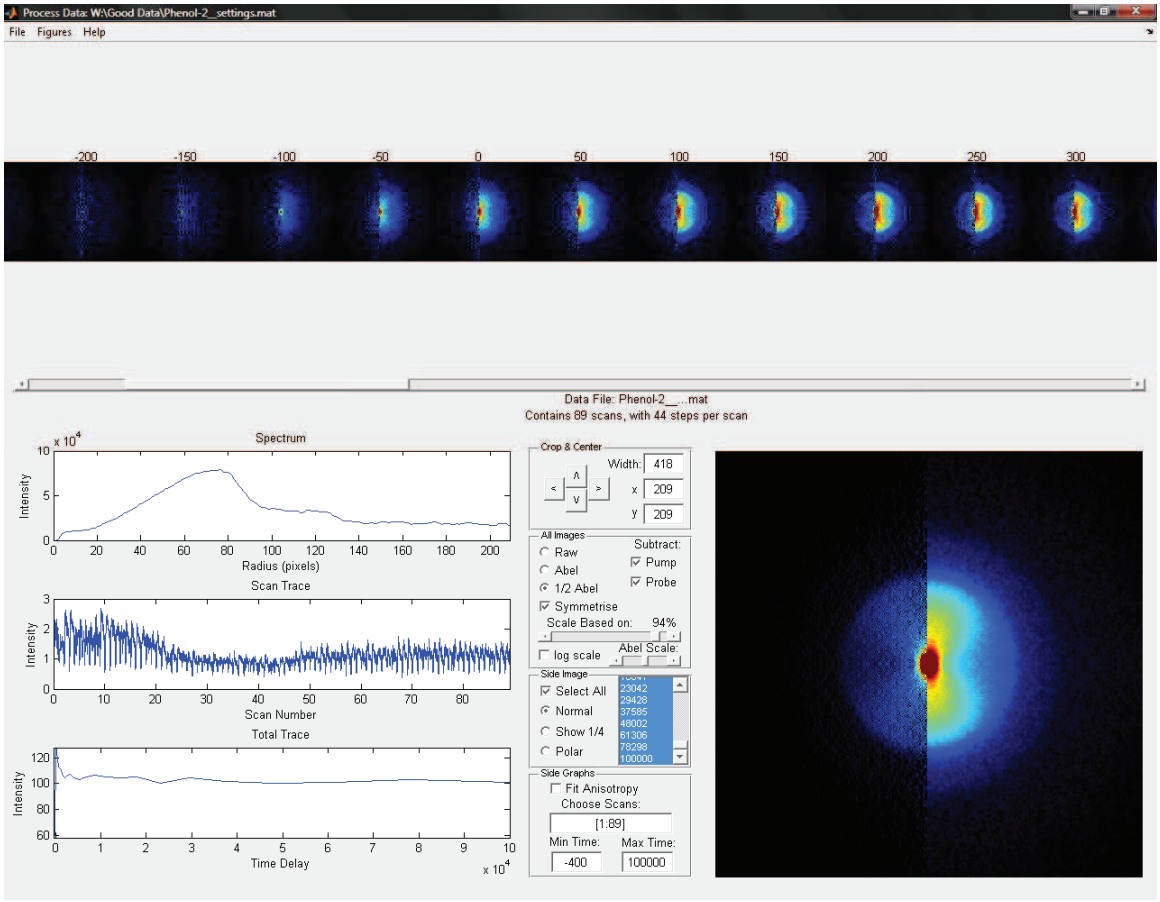


Figure 3.19: Process_Data user interface showing phenol data.

Once the data has been saved, there is a lot to do to the data in order to extract all the useful information. The processing and analysis of the data is not done in one step; two main programs are required to get the final results. The first of these programs, Process_Data (shown in Figure 3.19), performs the initial processing. It takes the raw data files from each scan, adds all the images with the same pump-probe delay together and subtracts the background. Any scans with unacceptable fluctuations due to laser pointing instability, sample running out, or other problems can be removed by the user at this stage. In addition it applies Abel transforms to the images using the matrix method described shortly, allows the user to change the centre point and, if required, symmetrise the image. Once the user is happy, it calculates intensity for each radius at each time delay. These intensities cannot be viewed in this program but are saved in a new file for viewing in Analyse_Data (Section 3.3.3).

Process_Data can accept data in several forms and add data sets together. It can also load up single images in a variety of formats. Once a single image is loaded it is

possible to load additional images to build up a time resolved scan, or add background images to remove background noise. If a standard scan file has been loaded, new scan files can be added. This allows the user to take several data runs using the same molecule and experimental conditions and add the data together. In order to do this, the two scans must have the same time steps. Often it is hard to define the point where the pump and probe pulse are exactly overlapped temporally. If care is not taken, this might mean that the time defined as zero (t_0) might actually be ± 50 fs. If this is the case, adding files together will effectively result in a lower time resolution. To avoid this problem, data sets should be compared with each other before addition to ascertain if an unacceptable time difference exists.

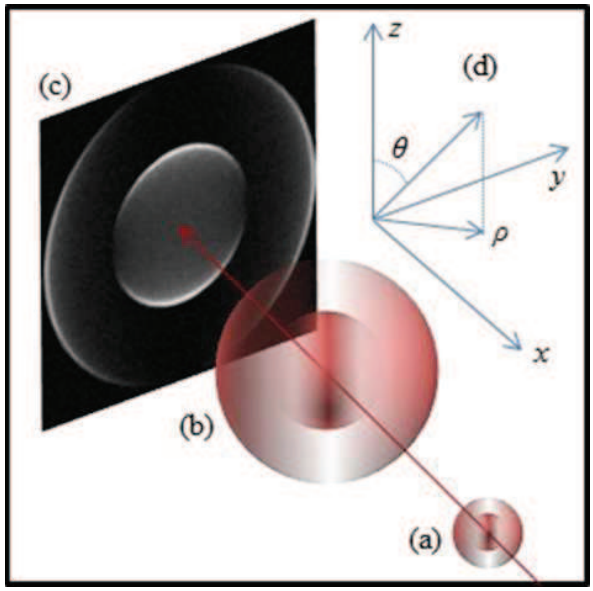


Figure 3.20: Cartoon demonstrating coordinate system. (a) Initial 3D expanding cloud of electrons. (b) Cloud of electrons after travelling some distance down the flight tube. Note they form cylindrically symmetric concentric spheres. (c) The 2D projection of the electron cloud onto the detector. (d) The co-ordinate system used in the text; x is the flight axis (also shown by the red arrow), and z the axis of symmetry.

Because the 3D charged particle distribution is projected onto the 2D detector in velocity map imaging techneques, some form of data processing is required to extract the original, cylindrically symmetric, 3D distribution from the 2D image (see Figure 3.20), and then calculate the velocity and anisotropy distributions associated with a photoionization or photodissociation event. In our data processing we employ the following matrix inversion method, which is similar to that previously employed by Cho and Na within the field of plasma diagnostics [26], and was developed for photoelectron imaging by Benjamin Sussman [27]. The cylindrically symmetric 3D charged particle distribution can be denoted by $I(x,y,z) \equiv I(\rho,z)$ (see Figure 3.20). The projection of this distribution onto the *y,z* (detector) plane can be written as:

$$P(y,z) = \int I(x,y,z)\, dx \qquad (3.14)$$

Using cylindrical coordinates, the cylindrical radius is $\rho^2 = x^2 + y^2$ and so:

$$P(y,z) = \int_y^\infty \frac{I(x,y,z)\,\rho d\rho}{\sqrt{\rho^2 - y^2}} \tag{3.15}$$

To obtain the original $I(x,y,z)$ from the measured $P(y,z)$ the inverse Abel transform may be used:

$$I(\rho,z) = -\frac{1}{4\pi}\int_r^R \frac{P(y,z)}{\sqrt{y^2-\rho^2}}\,dy \tag{3.16}$$

$I(\rho,z)$ has been obtained by numerous different strategies, as is discussed in more detail in Section 1.3.4. Most of the inversion techniques, including explicit evaluation of Equation 3.16, have the disadvantage of being computationally demanding and thus taking too long to be used in real time as data is being acquired. A great deal of optimization work has been done in the field of matrix inversion, and this can be utilised to speed up the Abel inversion process as follows. We begin by considering a particular slice z through the centre of the 3D distribution $I(x,y,z)$. This can be transformed into a matrix by partitioning the x,y-plane using lines of constant ρ and y, as $I(\rho,y) \rightarrow \mathbf{I}_{ij}$. This is illustrated in Figure 3.21; i denotes the maximum radius of the segment and j denotes the maximum y value of the segment. The area of each segment is denoted A_{ij}. The areas of the segments touching the x axis (axis perpendicular to imaging plane, see Figure 3.20) are given by:

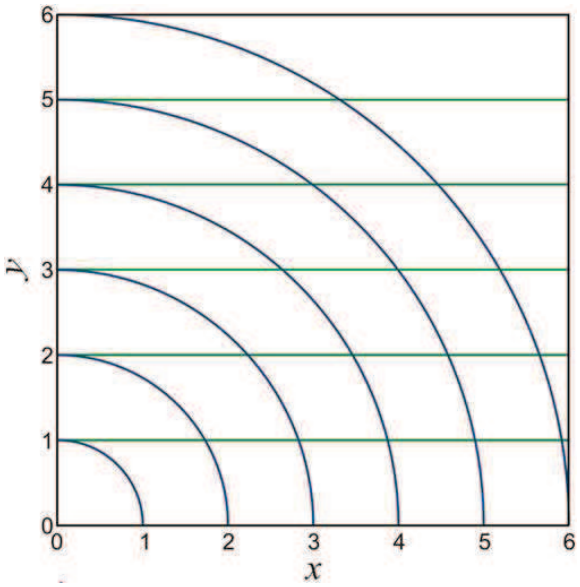


Figure 3.21: Partitioning of the plane for Abel inversion. Lines of constant radius(ρ) and y segment the plane with the area of each segment being A_{ij}. Please note the x,y plane shown is perpendicular to the original 2D projection $P(y,z)$. From [27].

$$A_{ij} = \int_{i-1}^{i} \sqrt{i^2 - y^2}\,dy = \frac{i}{2}\sqrt{-1+2i}\,i + \frac{1}{2}\sqrt{-1+2i} - \frac{i^2}{2}\arcsin\left(\frac{i-1}{i}\right) + \frac{\pi\,i^2}{4} \tag{3.17}$$

and all other segments are given by:

$$A_{ij} = \int_{i-1}^{i} \sqrt{j^2 - y^2} - \sqrt{(j-1)^2 - y^2}\, dy$$
$$= -\left(\frac{j^2}{2} - j + \frac{1}{2}\right)\arcsin\left(\frac{i}{j-1}\right) + \left(\frac{j^2}{2} - j + \frac{1}{2}\right)\arcsin\left(\frac{i-1}{j-1}\right)$$
$$+ \frac{i}{2}\sqrt{j^2 - 2j + 2i - i^2} - \frac{i}{2}\sqrt{j^2 + 2i - 1 - i^2} - \frac{1}{2}\sqrt{j^2 - 2j + 2i - i^2}$$
$$+ \frac{1}{2}\sqrt{j^2 + 2i - 1 - i^2} + \frac{i}{2}\sqrt{j^2 - i^2} - \frac{j^2}{2}\arcsin\left(\frac{i}{j}\right)$$
$$- \frac{i}{2}\sqrt{j^2 - 2j + 1 - i^2} \qquad (3.18)$$

The projection of the 3D distribution onto the imaging plane at large x is then simply $\mathbf{P} = 2\,\mathbf{A}\,\mathbf{I}$ and hence the inverse Abel transform is given by:

$$\mathbf{I} = \frac{1}{2}\mathbf{A^{-1}P} \qquad (3.19)$$

I, **A** and **P** are all $n \times n$ matrices, where n is the diameter of the image, and all have the same indices, i and j, corresponding to ρ and y. Note that the value of $\mathbf{I}_i$ represents the average value within the radial segment and not the value at $\rho = i$. However it is reasonable to approximate the average value to the central value by taking the correspondence $\rho = i$ - 0.5.

The main advantage of Equation 3.19 is that there are numerous rapid matrix inversion techniques available. This makes matrix inversion potentially suitable for many real time imaging solutions. As a rough guide to performance, we are able to invert one hundred 300 × 300 pixel images in one second using standard matrix inversion routines available in MATLAB on a computer with 2.67 GHz processer. Inversion times for different image sizes are shown in Figure 3.22 and it is interesting to note that the time increases with the cube of the size. To the best of our knowledge, this technique is approaching two orders of magnitude speed improvement over other methods. For example, the recent

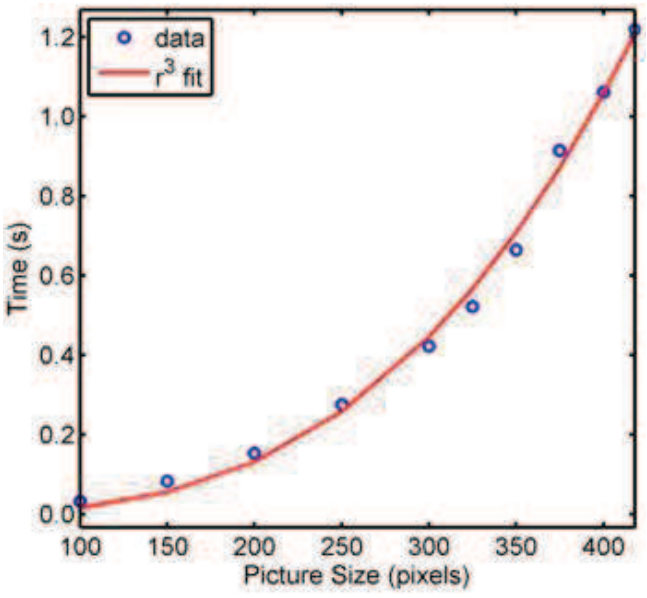


Figure 3.22: Time taken to Abel invert 44 images (an average data set) as a function of image size. Note the ρ^3 dependence.

polar onion peeling method approach developed by Roberts et al. [28] quotes a processing time of 0.6 seconds for a single 256 × 256 image on a 1.7 GHz PC. This in itself represents a speed improvement over other methods. In addition, least squares constraints on the solution may be easily imposed (for example, to demand a positive solution everywhere), and routines to perform this operation are readily available in commercial software. Unfortunately such constraints add significant additional complexity and much increased processing time (up to around 3 orders of magnitude slower). Given that the main appeal of the matrix inversion method is its rapid inversion speed, this is only of use with low signa-to-noise images.

In Figure 3.23 we present a comparison of the effect of background noise on the final result of the matrix inversion. A 500 × 500 pixel synthetic image test was created, with five rings of equivalent intensity but varying anisotropy. The image with no noise added was designed to be the same as the "cFew" image used by Whitaker and co-workers in their detailed comparison of several different (older) reconstruction methods [29]. Table 3.2 summarizes the results of conducting the same series of tests as Whitaker conducted on the cFew image. In all the aspects tested the matrix inversion

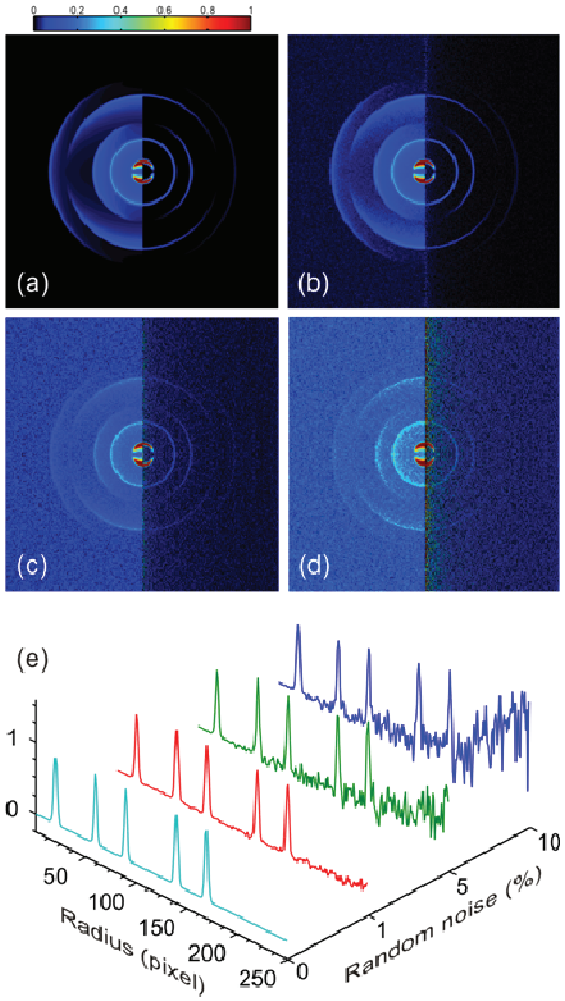


Figure 3.23: Effect of noise on Abel inverted images. (a) Synthetic test image comprising 5 rings with radii of 20, 60, 90, 140 and 170 pixels and associated anisotropy parameters of β_2 = 2, 0, -1, 2 and 1 respectively. The peak width is 2 pixels and the peak heights are all equivalent. The cylindrical axis of symmetry lies in the vertical direction. The left half of the image is the raw data, and the right hand side is the result after passing through the matrix inversion method described in the main text. (b) The same image with 1% random background noise added, raw image on left, inverted on right (c) Same as (b) but with 5% noise. (d) Same as (b) but with 10% noise. (e) The reconstructed distributions from each image shown above.

Speed (pix.)	Centre deviation (pix.)	Peak width (pix.)	Peak height	Peak area	Branching ratio (%)	β deviation
20 (β = 2)	0.25	3.10	0.93	3.596	19.10	0.027
60 (β = 0)	0.31	3.08	0.98	3.771	20.02	0.029
90 (β = -1)	0.35	3.06	0.99	3.780	20.17	0.000
140 (β = 2)	0.48	3.09	0.99	3.832	20.36	0.017
170 (β = -1)	0.50	3.06	1.00	3.832	20.35	0.000

Table 3.2: Results of performing a series of reconstruction tests, as detailed in [38], on the image shown in Figure 3.19(a) using the matrix method described in this work.

method appears to compare favourably with the other reconstruction methods. Due to lack of information as to the nature of the noise added to Whitaker synthetic images, no direct comparison of noisy data was possible, however Figure 3.23 demonstrates the effect that varying the level of noise has on the reconstructed image and on the resultant velocity distributions, and Figure 3.24 demonstrates with much lower levels of noise that the matrix inversion's treatment of noise is comparable to the onion peeling method. In Figure 3.24a, we present a synthetic image with 5 rings of varying intensities and anisotropies with random noise added. The result of the matrix Abel inversion method described above is shown in Figure 3.24b, and the results of the "Glass Onion" [30] onion peeling method is shown in Figure 3.24c. The reconstructed velocity distributions from both methods (Figure 3.24d) are a clear demonstration that the two methods yield equivalent results.

We can conclude from these tests that the matrix inversion method, whilst being much faster than other inversion methods, is comparable in output quality with most other inversion approaches. This method does have the limitation that it introduces centre-line noise and so is not ideal for very high resolution studies, however it is ideal for time resolved studies, where there are a great many images to process and the resolution is limited by the laser bandwidth.

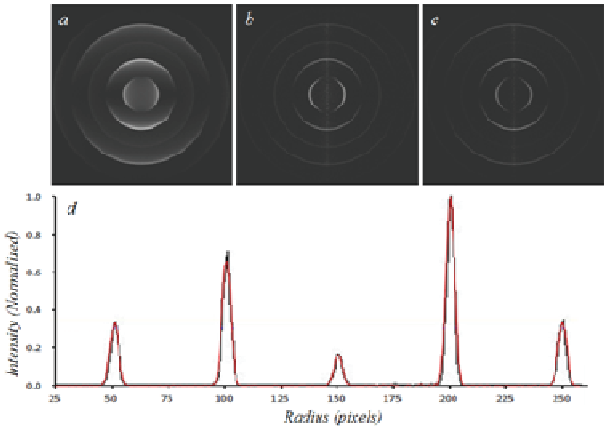


Figure 3.24: Abel Transform Comparison. a: original image b: matrix inversion method; c: onion peeling method; d: Velocity Distributions (red=matrix inversion, black=onion peeling).

3.3.3. Data Visualisation & Analysis (Analyse_Data)

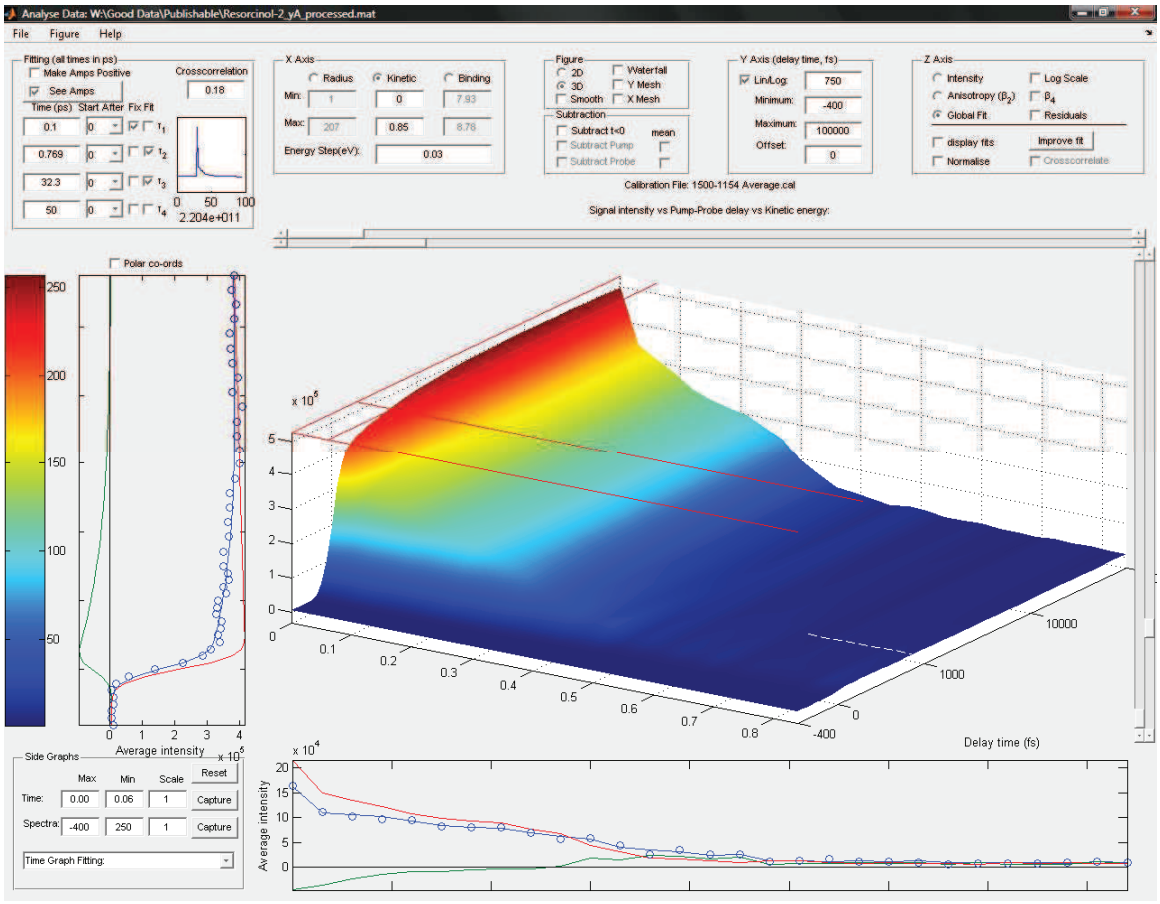


Figure 3.25: Analyse_Data User Interface during Global Fitting. The sliders at the sides allow regions of the 3D graph (indicated by the straight red lines) to be selected for displaying in the two side graphs.

The Analyse_Data program has four main purposes. The first is to convert the data from units of radius to energy (see Figure 3.26a); the second to visualise the data and provide the user with an intuitive display of the results of the experiment, the third is to fit time decays to the data and extract time constants (as seen in Figure 3.25) and the fourth is to fit anisotropy parameters (see Figure 3.26b), visualise how the anisotropy changes over time and energy, and fit time decays to the anisotropy data.

In order to convert the radial data into kinetic energy, a calibration file must be loaded. The calibration file is created from a calibration image (see the end of Section 3.2.3 for information of how calibration images are collected and processed) and contains all the variables required to scale a radius value to an energy value. The equations used to transform between energy and radius are:

$$I(E,t) = \sum_{i=r_1}^{r_2} I(r_i,t) \tag{3.20}$$

$$r_1 = \sqrt{A \cdot (E + \Delta E - E_0)} + r_0 \quad (3.21)$$

$$r_2 = \sqrt{A \cdot (E - E_0)} + r_0 \quad (3.22)$$

$I(E,t)$ is intensity as a function of energy and time, $I(r_i,t)$ is intensity as a function of radius and time, E is electron kinetic energy, r is the radial position on the image, A is the conversion factor between radius and energy, E_0 is the minimum known electron energy, r_0 is the radial position of that energy and ΔE is the width of energy bin (user defined). A, E_0 and r_0 are all defined in the calibration file. Using this relationship, each radial position is allocated to an energy bin, and the $I(t)$ of that bin is the sum of all the $I(r,t)$ allocated to the bin.

A single calibration file can be used to convert many different data sets. Care should be taken to create new files periodically and especially after any significant changes to the vacuum chamber or velocity mapping voltages, in order to ensure accuracy.

The data set can be visualised in several different ways. The data creates a 3D surface of time *vs.* energy *vs.* intensity. Analyse_Data applies a contour colour-map that can be edited for aesthetics or to emphasise certain features. The data can be viewed in 2D, where the only guide to intensity is the colour-map, or in 3D, with the intensity on the vertical axis. The intensity colour-map and height can be displayed on a logarithmic scale to emphasise low intensity features. The time axis can also be displayed on a linear/logarithmic scale (e.g. Figure 3.25), where the negative and small time delays are displayed on a linear scale and long time delays on a logarithmic scale. This helps to

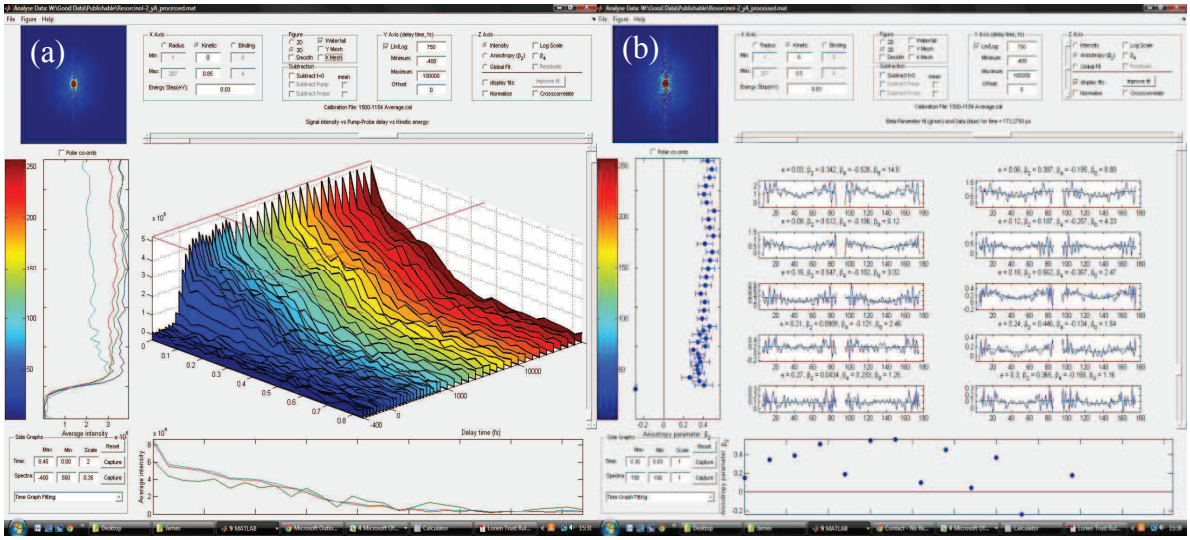


Figure 3.26: More screenshots of Analyse_Data with the same dataset as shown being fitted in Figure 3.25. (a) Raw data after being transformed into energy resolution (showing optional waterfall style 3D plotting). (b) The interface displaying a selection of individual anisotropy fits.

clearly display processes occurring on very different timescales within the same graph. The *x* axis can be displayed as radius, kinetic energy (i.e. the energy of the electrons, Equations 3.20 - 3.22), or binding energy (i.e. the energy left behind in the molecule). Binding energy, E_b, is calculated using:

$$E_b = \frac{hc}{\lambda_{pump}} + \frac{hc}{\lambda_{probe}} - E_k \tag{3.23}$$

The binding energy scale is advantageous when comparing different results from the same molecule taken at different wavelengths, as the scale is no longer dependent on the photon energy imparted to the molecule. In addition to the 3D graph showing all the data on the user interface, there are two line graphs that sum together all the data between user defined points (straight red lines on Figure 3.25) to show integrated time *vs.* intensity and energy *vs.* intensity graphs. These graphs are a useful comparison aid between different sections of the same dataset. All the graphs on the user interface can be easily exported into a publication standard image files, examples of which are seen throughout this thesis (e.g. Figure 3.27), or the data can be saved to a text file to enable the user to import to a spreadsheet.

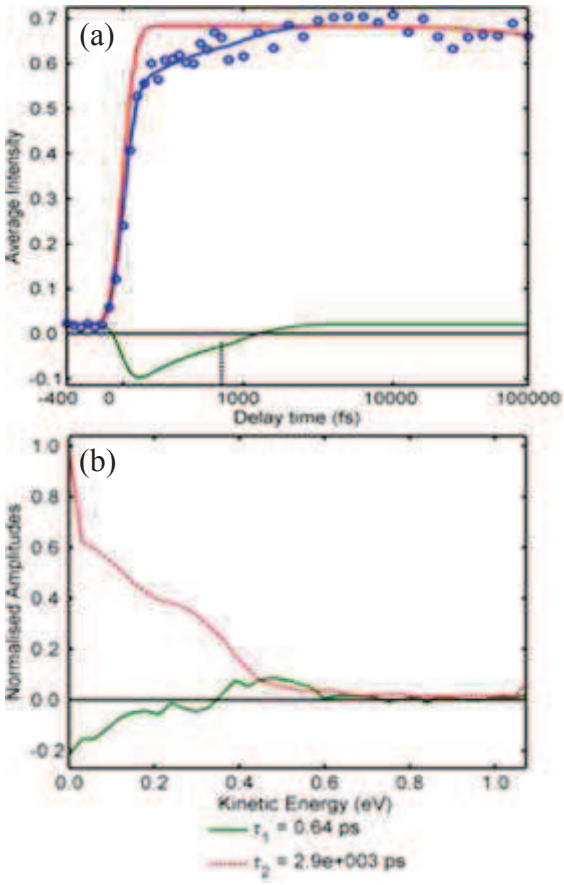


Figure 3.27: Resorcinol data fit using Equation 3.17. Graph (a) is the fit at energy = 0 eV. The blue circles are the data points, blue line is the fit, green and red lines are the two processes required to make the fit. Graph (b) is the Amplitudes $A_i(E)$ for all energy slices. Note the negative amplitude used to fit the rise. A discussion of the data can be found in Chapter 4.

Fitting the data to obtain time scales for the different processes involved is vital to gain a good understanding of the molecular dynamics involved. The 3D dataset is globally fitted at all photoelectron energies and all time delays simultaneously (see Figure 3.25 and Figure 3.27) using the following equation:

$$I(E,t) = \sum_{i=1}^{n} A_i(E)(e^{-\frac{t}{\tau_i}} \otimes \mathrm{cc(t)}) \tag{3.24}$$

where

$$cc(t) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{2\sqrt{\ln(2)}\ t}{\tau_{cc}}\right)\right) \tag{3.25}$$

n is the number of exponential decays used to fit the data; τ_i is the time constant for each decay; $A_i(E)$ is the amplitude of each decay required to fit each energy bin, $cc(t)$ is the rise due to the cross-correlation time τ_{cc} of the spectrometer which is modelled by a cumulative Gaussian. The factor of $2\sqrt{2\ \ln(2)}$ is to convert a Gaussian standard deviation to a full width half maximum. This standard fitting method returns a time constant and a set of amplitudes for each time process occurring. The amplitudes tell us at what electron energies the process is most dominant, and the time constant tells us the relaxation speed of the process. It is up to the user to decide how many time constants to fit. Care must be taken in choosing the number of exponentials to fit, and generally the fewest exponentials required to obtain a reasonable fit should be used. For this reason, fitting the data is much easier with an understanding of the excited states of the molecule and the possible photo-relaxation processes. For example in Figure 3.27, the initial energy bin (E = 0 eV) is shown (Figure 3.27a), along with the amplitudes for every energy bin (Figure 3.27b). It can be seen from the first graph that two decays are required to fit the data, a fast (green) one that rises up with time, and a very long lived

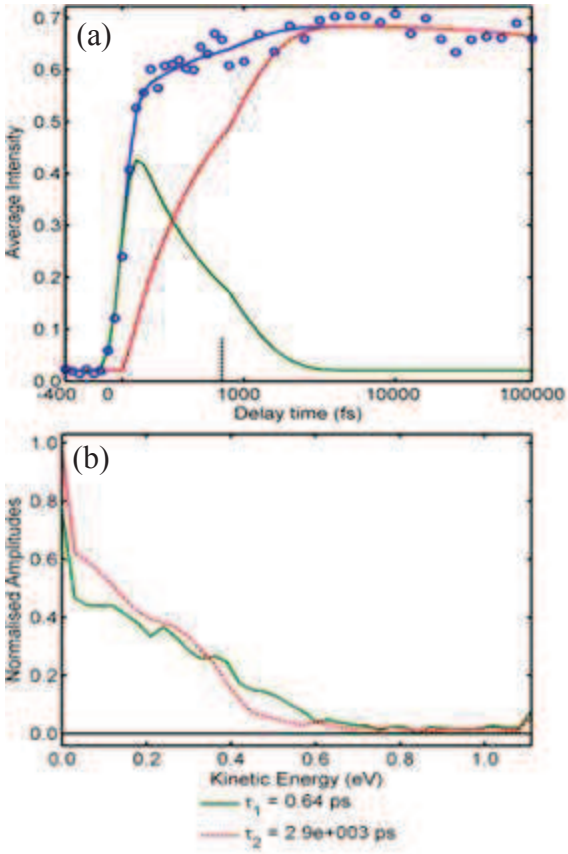


Figure 3.28: Data fit using the Consecutive Model. Graph (a) is the fit at energy = 0 eV. The blue circles are the data points, blue line is the fit, green and red lines are the two processes required to make the fit. Graph (b) is the Amplitudes $A_i(E)$ for all energy slices.

(red) one. This is shown on the amplitude graph as a small negative amplitude (i.e. exponential rise) for the green trace at $E = 0$ eV, and a larger positive amplitude for the red one. If one electronic state relaxes to another electronic state, then the second electronic state will have a slower rise time, on the same time scale as the decay of the first state. If fitting with the standard method, this results in negative amplitudes for the first state. In order to better fit data, a consecutive fit method has also been implemented (Figure 3.28). In this method the user decides if each process originates from zero or comes from another process. The data is then fitted using:

$$I(E,t) = \sum_{i=1}^{n} A_i(E)(e^{-\frac{t}{\tau_i}} \otimes r_i(t) \otimes \mathrm{cc(t)}) \tag{3.26}$$

where

$$r_i(t) = \begin{cases} 1 - e^{-\frac{t}{\tau_j}} \\ 1 \end{cases} \tag{3.27}$$

$r_i(t)$ is 1 when the process is originating from zero, and rises up with τ_j when it is not. τ_j is the time constant of the state the process is originating from. This method removes any artefacts caused by using an incorrect model; however care must be taken in deciding which state originates from which. The standard model is powerful in that it does not require any prior assumptions to reveal the dynamics. The consecutive model should only be used once the standard model has revealed evidence of consecutive dynamics between two or more states. In addition to these methods, the data can be exported to be fitted by an external program written by Helmut Satzger [31] if required. This allows the user to compare different fits and gives an additional measure of the fit reliability.

Finding the anisotropy of the data is a powerful tool to extract extra information about the dynamics of a molecular process. Often, different processes will give electrons with similar kinetic energies or similar decay times. This can cause difficulties in understanding the dynamics. Watching how the anisotropy changes with time and energy can give more information about the underlying processes. The anisotropy is often an indicator as to the electron orbitals involved and can help to identify states in complex molecular problems. To extract the anisotropy from an image [32, 33] when ionised using two photons of linear polarisation, the intensity as a function of radius and angle can be fitted with:

$$I(r,\theta) = \frac{\beta_0(r)}{4\pi}\left(1 + \beta_2(r)\,P_2(\theta) + \beta_4(r)\,P_4(\theta)\right) \quad (3.28)$$

where β_n is the n^{th} order anisotropy parameter, and $P_n(\theta)$ is the n^{th} order Legendre polynomial. We have previously discussed transforming between radius and energy values. If an image at each time delay is re-scaled onto an energy axis, then Equation 3.28 can be re-written in terms of energy and time:

$$I(E,t,\theta) = \frac{\beta_0(E,t)}{4\pi}\left(1 + \beta_2(E,t)\,P_2(\theta) + \beta_4(E,t)\,P_4(\theta)\right) \quad (3.29)$$

this equation can be used to describe all of the photoelectron data acquired. Fitting the data using this equation allows anisotropy parameters to be extracted for every energy and time delay.

Once the anisotropy has been fitted using nonlinear least squares method, the anisotropy parameters can be viewed in several different ways. Since anisotropy parameters are calculated for every energy and time position, the anisotropy parameters can be viewed using the same user interface as the intensity data. The β_0 parameter is simply related to the intensity, so does not give any extra information. The β_2 and β_4 parameters can be viewed as 3D colour-mapped graphs. As the intensity falls to zero, the signal-to-noise ratio becomes much smaller and so the fit becomes less reliable. The 3D graphs can be displayed so that only the points with high intensity or only the points with good fitting statistics are displayed. The side graphs that display the integrated data display error bars, and can be fitted to extract time constants for the evolution of the anisotropy over time. See Figure 4.6 in the following chapter for an example of extracting decay times from anisotropy data.

3.3.4. Model Data Creation (ImageGenerate)

ImageGenerate is a program that creates a model data set, with added noise and different decay times for different rings. It was created to help test the different data processing and analysis programs. The program allows the user to add as many different rings as they wish. Each ring can be assigned a different well defined radius, amplitude, peak width, decay time, and β_2 parameter (β_4 is set to zero). The model images are created by first creating a 3D distribution using Equation 3.28 and:

$$\beta_0(x,y,z) = \frac{e^{-\left(\frac{r(x,y,z)-r_r}{w}\right)^2}}{r(x,y,z)^2} \tag{3.30}$$

where

$$\theta(x,y,z) = \tan^{-1}\left(\frac{\sqrt{(y-W)^2+(z-W)^2}}{x-W}\right) \tag{3.31}$$

and

$$r(x,y,z) = \sqrt{(x-W)^2+(y-W)^2+(z-W)^2} \tag{3.32}$$

Here r_r is the central radius of the ring, w is the peak width, and W is ½ of the width of the picture. Once this three dimensional distribution is calculated, the 2D image is created by integrating over the x axis. Noise, if required, is then added using a random number generator, and a resultant image can be seen in Figure 3.29. A slice through the centre of the 3D distribution can also be viewed, which is convenient to compare with an Abel inversion of the whole image. When the model data are exported, it creates many images, each corresponding to a time delay. This file can be opened up by Process_Data, and can be used to test the software to ensure that the anisotropy fitting, global fitting and intensity calculations are correct. This is helpful in testing Process_Data (including the matrix Abel transform previously mentioned) and Analyse_Data, as the image parameters that are extracted by these programs can be compared with the original parameters inputted into ImageGenerate.

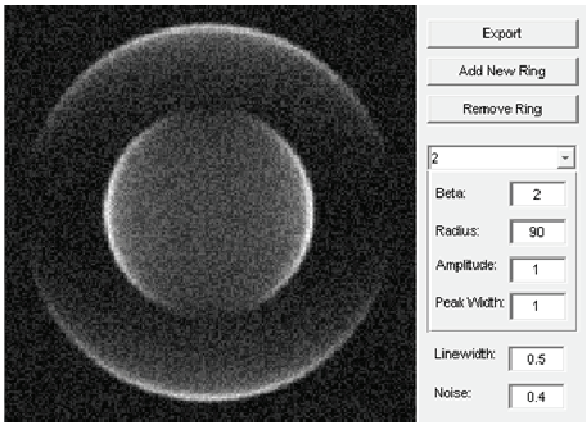


Figure 3.29: ImageGenerate User Interface.

3.4. Conclusion

The experimental setup and software as described in this chapter has been designed, assembled and commissioned over the course of my PhD; this process has formed the bulk of my project. It is now fully functioning and capable of efficiently collecting and analysing data. This setup is extremely versatile and able to provide insights into a wide range of chemical systems. The first complete experiment conducted on this apparatus is detailed in the following chapter. Other experiments are in process and the equipment and software should continue to provide and assist in analysing results for many more years.

3.5. References

[1] A. V. Smith and M. S. Bowers, *Phase Distortions in Sum-Frequency and Difference-Frequency Mixing in Crystals*, J. Opt. Soc. Am. B, **12**, 49, (1995).

[2] *SNLO Nonlinear Optics Code Available from A. V. Smith, AS-Photonics, Albuquerque, NM*

[3] Spectra-Physics, *Tsunami® Series Ti:Sapphire Ultrafast Oscillators*, DS 12072, (2008).

[4] Spectra-Physics, *Millenia® Pro S Series Green Lasers*, 000B 0224S, (2004).

[5] Spectra-Physics, *Spitfire® Pro XP Ultrafast Amplifier*, DS 01067, (2009).

[6] Spectra-Physics, *Empower® Q-Switched Green Lasers*, DS 011120, (2011).

[7] Spectra-Physics, *OPA-800C Ultrafast Optical Parametric Amplifier*, DS 11072, (2007).

[8] R. L. Fork, O. E. Martinez, and J. P. Gordon, *Negative Dispersion Using Pairs of Prisms*, Opt. Lett., **9**, 150, (1984).

[9] J. P. Gordon, R. L. Fork, and O. E. Martinez, *Negative Dispersion from Prisms*, J. Opt. Soc. Am. B, **1**, 437, (1984).

[10] O. E. Martinez, *3000 Times Grating Compressor with Positive Group-Velocity Dispersion - Application to Fiber Compensation in 1.3-1.6 μm Region*, IEEE. J. Quantum. Elect., **23**, 59, (1987).

[11] O. E. Martinez, J. P. Gordon, and R. L. Fork, *Negative Group-Velocity Dispersion Using Refraction*, J. Opt. Soc. Am. A, **1**, 1003, (1984).

[12] M. Rosete-Aguilar, F. C. Estrada-Silva, N. C. Bruce, C. J. Roman-Moreno, and R. Ortega-Martinez, *Calculation of Temporal Spreading of Ultrashort Pulses Propagating through Optical Glasses*, Rev. Mex. Fis., **54**, 141, (2008).

[13] R. Szipocs, K. Ferencz, C. Spielmann, and F. Krausz, *Chirped Multilayer Coatings for Broad-Band Dispersion Control in Femtosecond Lasers*, Opt. Lett., **19**, 201, (1994).

[14] O. E. Martinez, *Grating and Prism Compressors in the Case of Finite Beam Size*, J. Opt. Soc. Am. B, **3**, 929, (1986).

[15] E. B. Treacy, *Optical Pulse Compression with Diffraction Gratings*, IEEE. J. Quantum. Elect., **5**, 454, (1969).

[16] A. Stingl, C. Spielmann, F. Krausz, and R. Szipocs, *Generation of 11-fs Pulses from a Ti-Sapphire Laser without the Use of Prisms*, Opt. Lett., **19**, 204, (1994).

[17] W. Sellmeier, *Zur Erklärung der abnormen Farbenfolge im Spectrum einiger Substanzen*, Ann. Phy. Chem., **219**, 272, (1871).

[18] A. Stolow, *Minimum Profile Ultrahigh Vacuum Gate Valve Based on Linear / Rotary Motion Feedthrough*, J. Vac. Sci. Technol. A., **14**, 2669, (1996).

[19] U. Even, J. Jortner, D. Noy, N. Lavie, and C. Cossart-Magos, *Cooling of Large Molecules Below 1 K and He Clusters Formation*, J. Chem. Phys., **112**, 8068, (2000).

[20] A. T. J. B. Eppink and D. H. Parker, *Velocity Map Imaging of Ions and Electrons Using Electrostatic Lenses: Application in Photoelectron and Photofragment Ion Imaging of Molecular Oxygen*, Rev. Sci. Instrum., **68**, 3477, (1997).

[21] W. S. Hopkins, S. M. Hamilton, P. D. McNaughter, and S. R. Mackenzie, *VUV Photodissociation Dynamics of Diatomic Gold, Au_2: A Velocity Map Imaging Study at 157 nm*, Chem. Phys. Lett., **483**, 10, (2009).

[22] Y. Ogi, H. Kohguchi, D. Niu, K. Ohshimo, and T. Suzuki, *Super-Resolution Photoelectron Imaging with Real-Time Subpixelation by Field Programmable Gate Array and Its Application to NO and Benzene Photoionization*, J. Phys. Chem. A, **113**, 14536, (2009).

[23] E. Wrede, S. Laubach, S. Schulenburg, A. Brown, E. R. Wouters, A. J. Orr-Ewing, and M. N. R. Ashfold, *Continuum State Spectroscopy: A High Resolution Ion Imaging Study of IBr Photolysis in the Wavelength Range 440-685 nm*, J. Chem. Phys., **114**, 2629, (2001).

[24] D. Townsend, M. P. Minitti, and A. G. Suits, *Direct Current Slice Imaging*, Rev. Sci. Instrum., **74**, 2530, (2003).

[25] B. Y. Chang, R. C. Hoetzlein, J. A. Mueller, J. D. Geiser, and P. L. Houston, *Improved Two-Dimensional Product Imaging: The Real-Time Ion-Counting Method*, Rev. Sci. Instrum., **69**, 1665, (1998).

[26] Y. T. Cho and S. J. Na, *Application of Abel Inversion in Real-Time Calculations for Circularly and Elliptically Symmetric Radiation Sources*, Meas. Sci. Technol., **16**, 878, (2005).

[27] B. J. Sussman, *Quantum Control Using the Nonresonant Dynamic Stark Effect*, Doctoral Thesis, **Queens University**, Kingston, (2007).

[28] G. M. Roberts, J. L. Nixon, J. Lecointre, E. Wrede, and J. R. R. Verlet, *Toward Real-Time Charged-Particle Image Reconstruction Using Polar Onion-Peeling*, Rev. Sci. Instrum., **80**, (2009).

[29] A. T. J. B. Eppink, S.-M. Wu, and B. J. Whitaker, in *Imaging in Molecular Dynamics: Technology and Applications* (B. J. Whitaker, ed.), Cambridge University Press, Cambridge, p. 65-112, (2003).

[30] S. Manzhos and H. P. Loock, *Photofragment Image Analysis Using the Onion-Peeling Algorithm*, Comput. Phys. Commun., **154**, 76, (2003).

[31] H. Satzger and W. Zinth, *Visualization of Transient Absorption Dynamics - Towards a Qualitative View of Complex Reaction Kinetics*, Chem. Phys., **295**, 287, (2003).

[32] K. L. Reid, *Photoelectron Angular Distributions*, Ann. Rev. Phys. Chem., **54**, 397, (2003).

[33] T. Suzuki, *Femtosecond Time-Resolved Photoelectron Imaging*, Ann. Rev. Phys. Chem., **57**, 555, (2006).

4. Excited State Relaxation Dynamics in Phenol, Catechol, Resorcinol and Hydroquinone

4.1 Introduction

In the previous chapter the experimental setup was discussed in some detail. In this chapter, the first experimental results obtained on this new setup are presented and discussed. The initial setup of the equipment was full of challenges. The time taken to collect the results that are presented here was relatively short, but this was made possible by many months of calibration, systematic resolution of teething problems and optimisation of equipment to allow for the efficient collection of data.

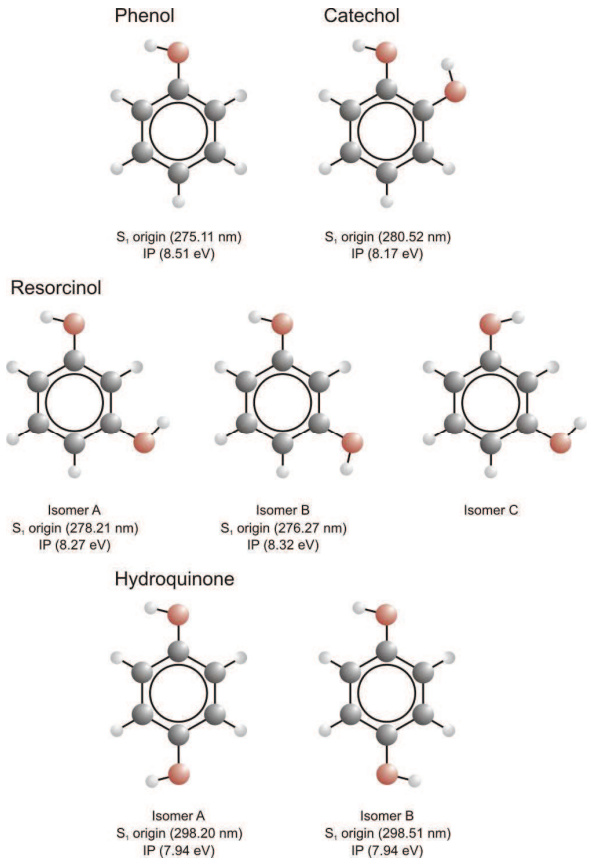


Figure 4.1: Schematic depiction of the four molecules studied in this chapter: phenol, catechol, resorcinol and hydroquinone. The conformers shown are those expected to be present in the molecular beam. The positions of the respective S_1 origins are given along with the adiabatic ionization potentials. Note that isomer C of resorcinol is predicted to exist but has not been observed spectroscopically.

The spectroscopy and dynamics of phenol and its derivatives (as seen in Figure 4.1) have received considerable attention over the last 25 years, both experimentally [1-14] and theoretically [15-25]. As discussed in Section 1.2.2, the role played in molecular dynamics by the optically bright ππ* (S_1) and the close lying optically dark πσ* (S_2) state, which is dissociative along the O-H stretching coordinate, is of particular interest. Such states of πσ* character are now widely recognised as playing a key role in the relaxation dynamics of many electronically excited species containing OH, NH and SH groups [19, 26].

In phenol the $S_2 \leftarrow S_0$ transition possesses little or no oscillator strength, although it may, in some instances, be accessed directly as a result of vibronic coupling. In addition, radiationless transfer to (and subsequently from) the S_2 state may also take place at conical intersections formed with other electronic states, providing significant mechanistic pathways for electronic relaxation. Ashfold and co-workers have brought significant new insight through comprehensive investigation of the photophysics of the phenol molecule [27-34]. Their investigations, spanning several years, use H-Rydberg atom photofragment translational spectroscopy and have identified two different photodissociation mechanisms for photoexcitation with wavelengths above and below a threshold of 248 nm. When the molecule is excited directly to the $\pi\pi^*$ state origin (275.11 nm) the H atom kinetic energy release distribution is bi-modal. A high energy component consists of a series of peaks, which can be correlated with vibrational levels of the ground state of the phenoxyl radical co-fragment. An unstructured low energy component is present at all excitation wavelengths and has been attributed to "statistical" H atom dissociation from a highly vibrationally excited ground state [10, 11, 19, 25, 33]. The origin of the high energy component has been the source of some discussion [1, 27, 33], but the most recent works by various groups [14, 25, 35] confirms that the dissociation comes from the H atom tunnelling under the potential barrier caused by the S_1/S_2 conical intersection, with subsequent dissociation along the S_2 surface, as initially proposed by Sobolewski *et al* [19].

The S_2 ($\pi\sigma^*$) state in phenol has been identified as having significant Rydberg (3s) character in the vertical Franck-Condon region. At more extended O-H distances the electron density evolves towards being localised on the OH group and exhibits a node along the O-H bond [19]. The S_1 and S_2 states have different symmetry and so should not undergo coupling. Vibrational modes of specific symmetry may however induce a vibronic interaction between the two potential surfaces. A detailed analysis of the energy disposal within the phenoxyl co-fragment by Ashfold and co-workers has provided strong evidence that odd quanta excitation in the υ_{16a} (a_2) ring torsion mode is responsible for mediating the S_1/S_2 interaction [27]. Recent work from Stavros and co-workers monitoring the time resolved appearance of H atom photoproducts has concluded that for excitation wavelengths with insufficient energy to populate $\upsilon_{OH} = 1$ in the S_1 state (> 250 nm), tunnelling always proceeds exclusively from the zero-point energy (ZPE) level of this mode. They observed that excitation of vibrational modes orthogonal to the O-H stretching coordinate (so-called "spectator modes") has no

significantly effect on the H atom tunnelling rate [14]. These authors also reported time-resolved decay traces of the phenol parent ion, observing a fall in the overall S_1 lifetime with increasing excitation energy. It was suggested that this may be due to a competing mechanism driven by the spectator modes involving internal conversion between S_1 and vibrationally excited S_0.

As the excitation wavelength is reduced below 248 nm, the intensity of the fast H atom signal begins to fade and a new, structured, fast H atom product channel begins to appear in its place. This new channel exhibits a considerably higher average kinetic energy and significant recoil anisotropy ($\beta \sim -0.5$). This observation has previously been attributed to S_1 being excited above the barrier formed by the S_1/S_2 conical intersection, followed by rapid internal conversion to the S_2 potential surface. This then leads to O-H bond fission on a timescale faster than the parent rotational period [33]. An alternative mechanism, however, invoking direct excitation to the S_2 state (facilitated by υ_{16b} (b_1) mediated vibronic coupling with the higher-lying S_3 ($\pi\pi^*$) state) has recently been proposed [25, 27].

Within the excitation range of 275 nm to 193 nm, the slow component present in the H atom kinetic energy release distribution has been attributed, in part, to a second conical intersection between the S_2 and S_0 potential surfaces at highly extended O-H distances. Subsequent intramolecular vibrational redistribution (IVR) then leads to "statistical" dissociation of the highly vibrationally excited ground (S_0) state, giving rise to $\widetilde{X}^2B_1$ phenoxyl radicals [10, 11, 33]. This interpretation is, however, not fully reconciled with time resolved H atom elimination studies of Stavros and co-workers, who observe both the fast and slow kinetic energy channels to appear on an extremely rapid (< 200 fs) timescale following excitation at 200 nm [12,13]. A second contribution to the slow H atom component observed by Ashfold's group is believed to arise from (unwanted) multiphoton absorption and subsequent decay of "superexcited" states [34, 36].

In contrast to phenol, the spectroscopy and dynamics of the excited electronic states of gas-phase catechol (1,2-dihydroxybenzene), resorcinol (1,3-dihydroxybenzene) and hydroquinone (1,4-dihydroxybenzene) have received comparatively little attention. UV absorption spectra of all three species, recorded in a cell at elevated temperature, were first reported by Beck in 1950 [37]. More recent work by Lubman and co-workers employed molecular beam methods to record (1+1) REMPI spectra close to the S_1 origin of all three molecules, observing well-resolved vibrational structure [38]. A

number of additional spectroscopic studies in the UV have subsequently been performed on the S_1 state in each of the catechol [39-42], resorcinol [43-46] and hydroquinone [45, 47-52] systems. In catechol, in the S_1 state, the free O-H bond lies out of the plane defined by the rest of the molecular framework by approximately 24^o [41]. In the ground state, however, a hydrogen bonding interaction between one of the OH groups and the adjacent O atom means that only a single, planar S_0 conformer is present in the gas-phase at room-temperature [53-55]. In contrast, both resorcinol and hydroquinone have two planar S_0 conformers that are present even in a jet cooled molecular beam, and two different S_1 origins are therefore observed (see Figure 1). As in the case of phenol, the S_1 excited state equilibrium geometries of resorcinol and hydroquinone are also found to be planar [43, 49].

Ashfold and co-workers have reported a very recent detailed study of the S_1 ($\pi\pi^*$)/S_2 ($\pi\sigma^*$) excited state electronic dynamics in catechol [56]. By employing a similar experimental methodology to their previous studies on phenol, this group obtained H Rydberg atom photofragment translational spectroscopy data at range of excitation wavelengths between 280.52 nm (the S_1 origin) and 193.3 nm. In instances when the excitation wavelength was less than 270 nm, a bimodal (and isotropic) H atom kinetic energy release distribution with a structured fast component was observed. This was interpreted in a manner similar to phenol, with the non-hydrogen bonded H-atom tunnelling under the S_1/S_2 barrier giving rise to the fast component. Analysis of the energy dispersal in the catechoxy radical co-fragment suggests that this process is vibronically mediated by ring-puckering modes. It was also noted in this study that the catechoxy radical product state distribution was highly sensitive to the degree of OH torsional excitation (of the non-H-bonded group) in the catechol S_1 state. The catechol S_1/S_2 conical intersection was suggested to be significantly lower than in phenol; this was principally attributed to a reduction of the S_2 vertical excitation energy resulting as a consequence of the large ($\geq$ 2500 cm^{-1}) decrease in O-H bond strength. This decrease arises both due to the electron donating properties of the additional *ortho*-substituted hydroxyl group, and due to the intramolecular hydrogen bonding interaction between one of the OH groups and the adjacent O atom. As the excitation wavelength is shortened below 270 nm, a broad but unresolved distribution of fast H atoms gradually evolves to higher average kinetic energy. The emergence of a competing H atom elimination mechanism involving direct excitation to the S_2 state was suggested to account for this observation.

This chapter lays out our recent investigations of the relaxation dynamics in gas-phase phenol, catechol, resorcinol and hydroquinone using time-resolved photoelectron imaging following excitation at 267 nm. This has recently been submitted for publication in the Journal of Chemical Physics [57]. We observe clear differences as well as some strong similarities in the temporal evolution of all four systems, with two clear dynamical timescales being apparent in all cases. What we observe in these photoelectron angular distributions provides additional mechanistic insight. The catechol system is of particular interest to this thesis as it is a sub-unit of the 5,6-dihydroxyindole molecule, which is discussed in Chapter 2 and 5. 5,6-dihydroxyindole is known to be an important building block in the eumelanin pigmentation system, the primary role of which is to protect the body from the potentially damaging effects of UV radiation. As such, this study complements the time-resolved work reported in Chapter 2 on gas-phase indole and 5-hydroxyindole. Dihydroxybenzenes are also key constituents of many other biomolecules, including various flavonoids and the hormones dopamine and adrenaline. Developing a more detailed general understanding of the photochemical behaviour of these systems is therefore of great importance.

4.2 Experimental Setup

The experimental setup used in this chapter has been explained in detail in chapter 3. Phenol, catechol, resorcinol and hydroquinone were purchased from Sigma-Aldrich and used without further purification. The phenol sample was held in the external gas pick-up cell to avoid melting the sample. All the other species were held in the cartridge within the valve body (for more details see Section 3.2). The Evan-Lavie [58] valve was usually opened for 30 μs. Helium at 3 bar pressure was used as a seeding gas for all the molecules, and the samples were internally and translationally cooled through supersonic expansion into the source chamber. A velocity mapping voltage ratio of 0.77 was used throughout. The maximum possible electron kinetic energy (E_{max}) resulting from the two photon resonantly enhanced ionisation process is related to the molecule's ionisation potential (IP) and the wavelength of the pump and probe photons (λ_{pump} and λ_{probe}) by the following equation:

	Purity (%)	Melting Point (°C)	*IP* (eV) [*source*]	E_{max} (eV)	Wavelength (nm)		$V_{repeller}$ (V)
					λ_{pump}	λ_{probe}	
Phenol	≥ 99.5	40.5	8.51 [85]	0.244	301.3	267.3	1000
Catechol	≥ 99	105	8.17 [41]	0.582	301.5	267.3	1500
Resorcinol	≥ 99	110	8.30 [45]	0.457	301.4	267.1	1500
Hydroquinone	≥ 99	172	7.94 [45]	0.690	309.1	268.4	1500

Table 4.1: Summary of some of the properties and experimental conditions used for the four molecules studied in this chapter. Samples were purchased from Sigma-Aldrich. Melting points were obtained from NIST webbook [59]. *Cut-off energies (E_{max}) were determined using equation 4.1 in text. Central wavelengths were determined using an Ocean Optics USB2000+ spectrometer. Repeller voltages were applied to the repeller plate as shown in Figure 3.12. Extractor voltages can be found by multiplying repeller voltages by 0.77.*

$$E_{max} = \frac{hc}{\lambda_{pump}} + \frac{hc}{\lambda_{probe}} - IP \tag{4.1}$$

where h is the Planck constant and c is the speed of light. From this value, the repeller voltage $V_{repeller}$ was chosen to obtain the best possible resolution without losing the most energetic electrons off the edge of the detector. The data was collected by scanning the stage to alter the time delay between the pump and the probe pulse. The steps taken were from -400 fs to +750 fs in 50 fs increments, followed by 20 exponentially increasing steps between +750 fs and +100 ps. Typically at each step an image was acquired for 5 seconds, and pump-alone and probe-alone background images were acquired for 2.5 seconds. For each molecule this scan was repeated around 100 times and the images added together to improve the signal to noise. Table 4.1 summarises the main properties and experimental treatments of the four molecules studied.

4.2.1 Photoelectron Images

In Figure 4.2, we display the series of photoelectron images resulting from (1 + 1′) ionisation of phenol, catechol, resorcinol and hydroquinone. 44 pump-probe delay time (Δt) images were collected for each molecule, but for clarity and conciseness only certain, selected, images are displayed here. These images were generated by subtracting the one-colour pump-alone and probe-alone background images from the raw pump-probe image at each time delay. Using the procedure outlined in Section 3.3, photoelectron spectra were generated from the images for each of the four molecules. An example of a matrix-inverted image from the phenol system is presented in Figure 4.5. The photoelectron angular distributions seen in all four molecules exhibit considerable anisotropy at all delay times, peaking in the vertical direction (along the laser polarisation axis).

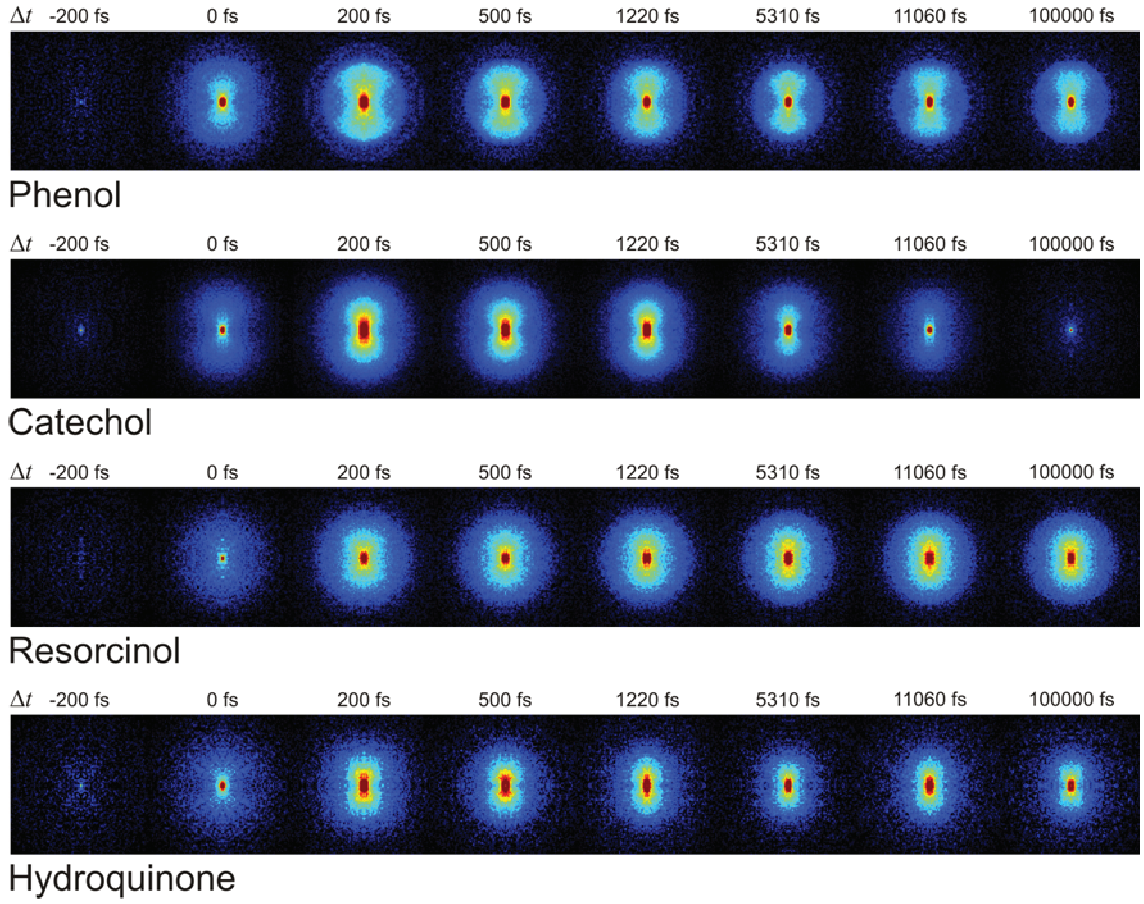


Figure 4.2: Photoelectron images for a series of selected pump-probe delay times for the four molecules investigated here. Pump alone and probe alone signals have been subtracted from the images and the images have been four-fold symmetrised. The seemingly unusual time steps for some images are due to the exponential stepping method for time longer time values.

4.2.2 Time-Resolved Photoelectron Spectra

After processing and background subtraction, the collected data sets for all four molecules are displayed in Figure 4.3. The energy axis is plotted in terms of electron binding energy, as the adiabatic ionisation potentials (IP) of each molecule has been previously reported using high-resolution techniques [41, 60, 61]. In order to clearly display both the short and long time dynamics, the data is displayed in a linear/logarithmic time axis. The electron energy is plotted in binding energy (E_b), as described in Section 3.3.3 and using:

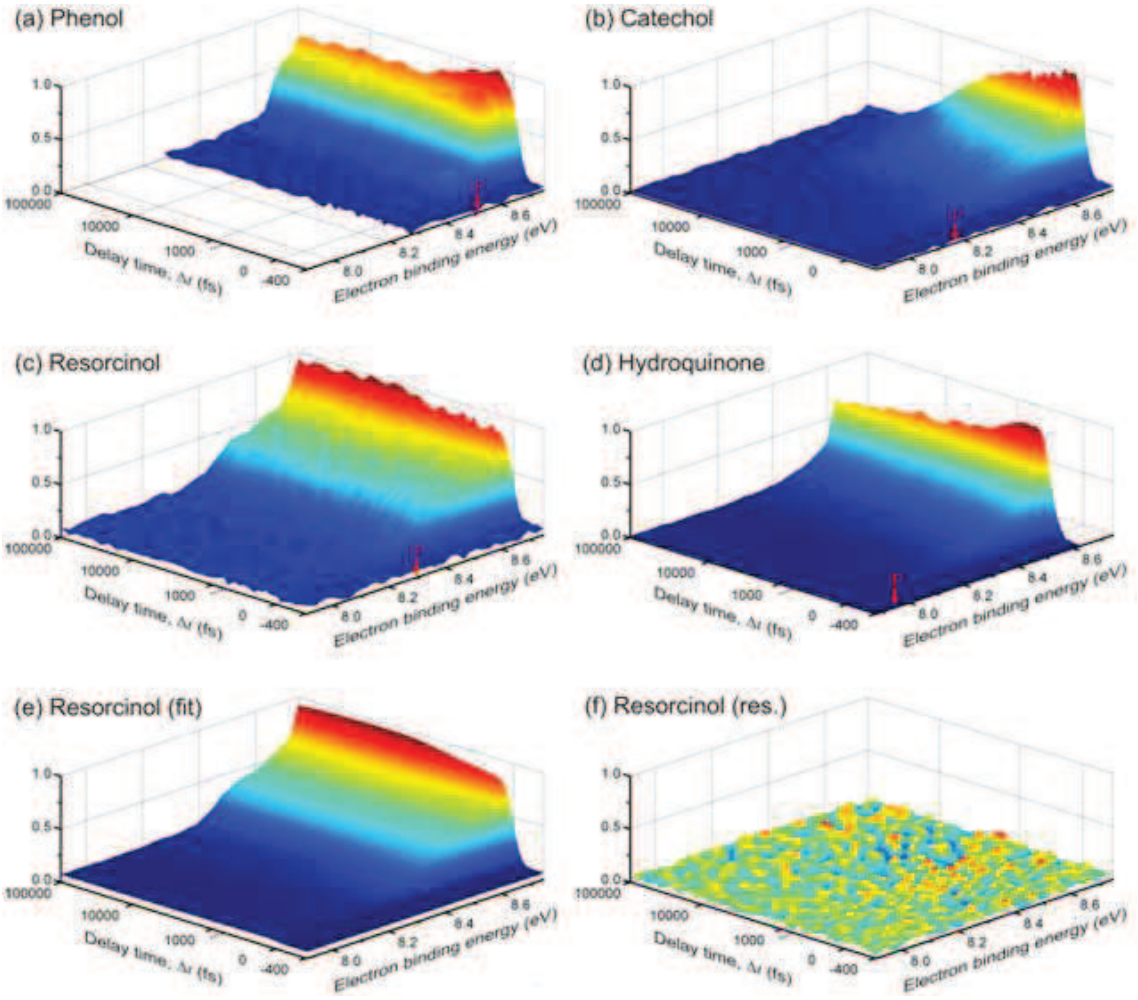


Figure 4.3: Time dependant photoelectron spectra of phenol (a), catechol (b), resorcinol (c) and hydroquinone (d). The time axis is plotted on a linear scale between -400 and +750 fs, and a logarithmic scale thereafter. The energy scale is plotted in binding energy with the ionisation potentials marked on the plots. The data is partitioned onto 0.3 eV energy bins. Also shown are the fit to the resorcinol data (e) and the associated residuals (fit – data) (f).

$$E_b = \frac{hc}{\lambda_{pump}} + \frac{hc}{\lambda_{probe}} - \frac{\rho^2}{A} \tag{4.2}$$

where ρ is the radius (in pixels) and A is the calibration constant, explained in Section 3.3.3, and collected using a Xenon photoelectron image. In all cases only the D_0 (π^{-1}) cation electronic state is energetically accessible [62, 63]. The time dependence of the data sets were analysed using the techniques described in section 3.1.3, and two exponential decays were extracted through global fitting with the following equation:

$$I(E,t) = A_1(E)(e^{-\frac{t}{\tau_1}} \otimes cc(t)) + A_2(E)(e^{-\frac{t}{\tau_2}} \otimes cc(t)) \tag{4.3}$$

The intensity I as a function of time t and energy E can be expressed as two exponential

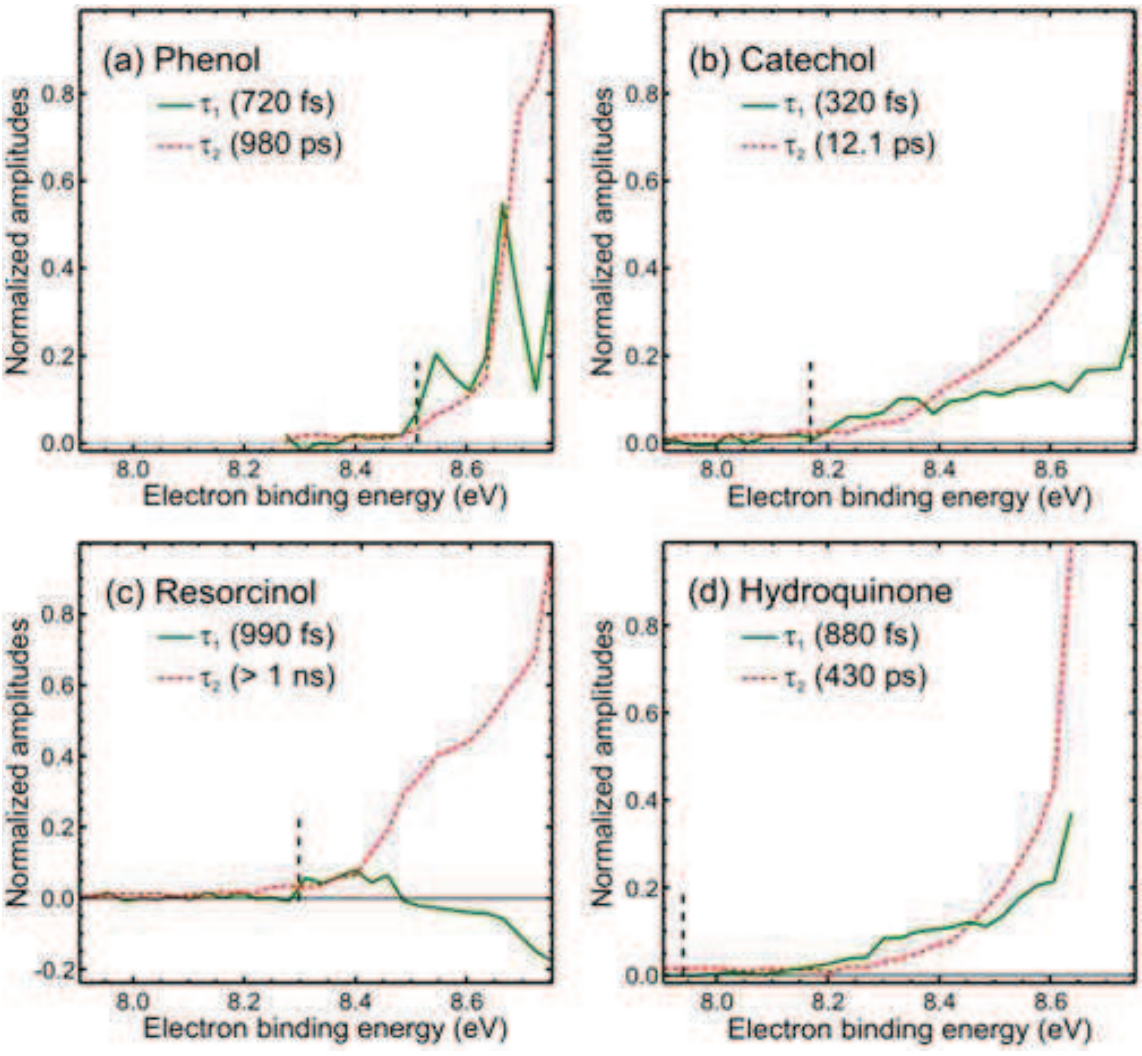


Figure 4.4: Decay associated spectra for phenol (a), catechol (b) resorcinol (c) and hydroquinone (d). Spectra were extracted from a global fit to the date displayed in Figure 4.3. Vertical dashed lines denote the adiabatic ionisation potential.

decays with time constants τ_1 and τ_2 convoluted with the instrument response (*cc*) (which is a 160 fs FWHM gaussian, measured as described in Section 3.3.1). If the time response of each energy value it treated separately, $A_i(E)$ is the amplitude of the exponential decay characterised by τ_i required to fit the data at energy E. This de-convolution of the data extracts not only the time scale of each process, but also the importance of that process across different regions of the spectrum. To illustrate the good quality of this model, Figure 4.3(e-f) show the corresponding fit and associated residuals for the resorcinol data. The decay associated spectra $A_i(E)$ for each molecule are displayed in Figure 4.4. The phenol spectrum appears to show some partially resolved structure at small Δt values.

Several observations can initially be drawn from the data. Firstly, there are no negative time (i.e. probe-pump) dynamics. This is due to the deliberate restriction of the probe pulse to wavelengths that have very low absorption by the molecule (see Table 4.1 for exact wavelengths). It can be seen in Figure 4.3 that the signal drops to zero at negative time delays. This gives us confidence that decay times extracted from our fitting techniques will not be distorted by probe-pump effects.

Secondly, there are two clear timescales involved in all the data sets. It is clear that in every case there is a fast process that occurs in less than a picosecond and a slower process that occurs over many picoseconds. Table 4.2 displays the time constants extracted by this method. There is a striking difference in the long-time decay exhibited by catechol when compared to phenol, resorcinol and hydroquinone, with catechol being considerably faster. In contrast to the other systems under study, the resorcinol

	Time Constants		
	τ_1 (fs) (±15%)	τ_2 (ps) (±15%)	τ_β (fs) (±20%)
Phenol	720	980	660
Catechol	320	12.1	720
Resorcinol	990	> 1000	930
Hydroquinone	880	430	480

Table 4.2: Summary of the time constants extracted by the global fit of the photoelectron spectra (τ_1 and τ_2) as discussed in this section, and the fit of the anisotropy data (τ_β), as discussed in the next section.

data shows a long-time signal that is clearly increasing towards extended pump-probe delays. In this case it was not possible to reliably extract a numerical value for τ_2 (given the range of pump-probe delay times sampled) and we can only quote a lower limit of 1 ns. The resorcinol data also shows a very striking negative amplitude component in the τ_1 DAS. This reveals that at least some of the signal associated with the long-lived decay (as described by the τ_2 DAS) must originate from a sequential process, as is also initially suggested by the rising photoelectron signal in the corresponding raw data shown in Figure 4.3(c). In all four systems the DAS associated with τ_1 and τ_2 span the same binding energy region, although the relative amplitudes are somewhat different.

Thirdly, it can easily be seen that there is no signal at energies below the ionisation threshold as indicated by red arrows in Figure 4.3. This gives us confidence that no three photon (1 + 2' or 2 + 1') processes are occurring. A previous time-resolved study of phenol reported by Schick and Weber observed the presence of "superexcited" states (i.e. neutral states lying above the adiabatic ionisation potential) that may be populated following two-photon absorption via the S_1 state at 275 nm [36]. The presence of superexcited states would be apparent in the form of signals extending to binding energies below the adiabatic IP. The fact that almost no photoelectron signals are observed in this region strongly suggests that these states are not a significant factor in any of our data.

A fuller discussion of the dynamical interpretation of the decay associated spectra and time constants can be found in Section 4.3.

4.2.3 Photoelectron Angular Distribution

The angular distribution of the photoelectron images can reveal extra information about the dynamics in the systems under study. In our case, with two photon ionisation using linear polarisation, we may express this distribution as a function of electron energy E and pump-probe delay time t, using the previously defined (Section 3.3.3) anisotropy parameters β_n and the Legendre polynomials $P_n(\theta)$.

$$I(E,t,\theta) = \frac{\beta_0(E,t)}{4\pi}\left(1 + \beta_2(E,t)\,P_2(\theta) + \beta_4(E,t)\,P_4(\theta)\right) \quad (4.4)$$

This equation can be used to fit the data and extract the relevant anisotropy parameters. To avoid distortions due to the vertical stripe of amplified noise (see Figure 4.5), the data was fitted over the angular range $5^o < \theta < 175^o$ and $185^o < \theta < 355^o$. It was found with all the data fitted here that β_4 was essentially zero. In regions of the photoelectron spectra where only short time dynamics (τ < 1 ps) are observed, β_2 was also essentially zero, i.e. the distribution is completely isotropic. This can be easily observed in the decaying isotropic outer ring in the Phenol data (Figure 4.2). Towards the centre of the images, where the longer lived time dynamics are predominant, β_2 was found to be non-zero, characteristic of a highly anisotropic distribution. At each time delay, β_2 was found to be largely invariant over this central, long lived, portion, although there was a clear evolution as the pump – probe delay increased from zero. This evolution was fitted using the following equation and the fit along with the β_2 values are plotted in Figure 4.6

$$\beta_2(t) = e^{-\frac{t}{\tau_\beta}} \quad (4.5)$$

For all four molecules the lifetime (τ_β) was found to be under a picosecond, and represented an increase in β_2 of up to 25%. This

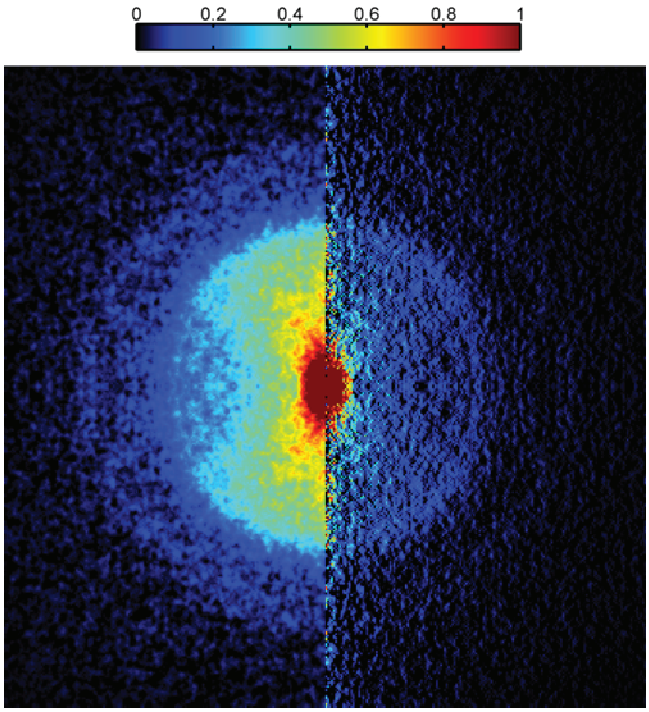


Figure 4.5: Photoelectron image of phenol recorded at 200 fs. The left half is exactly the same as shown in Figure 4.1. The right half is the same image after Abel inversion. The linear polarisation direction of both the pump and the probe beams is vertical with respect to the image.

evolution is plotted in Figure 4.6, which also shows the rising exponential fit to the data. The fitted lifetime τ_β ranged from 480 fs (hydroquinone) to 930 fs (resorcinol). Interestingly, in catechol and hydroquinone, the value of τ_β deviates considerably from the value of τ_1 obtained in the DAS fits to the photoelectron data shown in Figure 4.4. This deviation is too large to be simply attributed to the uncertainty present in the fitted time constants. Interestingly, in the case of catechol, τ_β is greater than τ_1 (720 fs *vs*. 320 fs) whereas in hydroquinone this situation is reversed (480 fs *vs*. 880 fs). This possibly points to a more complex dynamical picture than is suggested by the DAS fits alone and clearly demonstrates the additional insight that the highly differential nature of the photoelectron imaging approach provides.

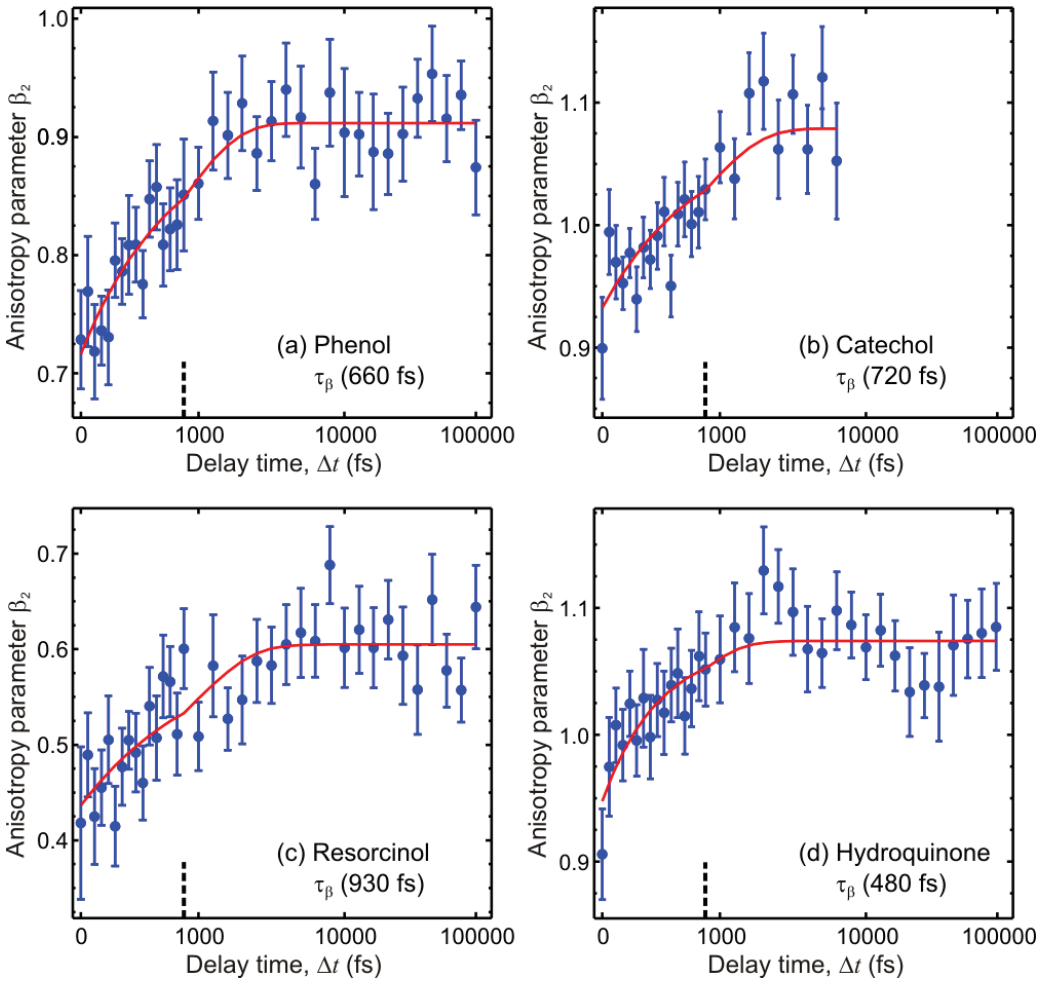


Figure 4.6: Extracted anisotropy parameter β_2 as a function of pump-probe delay time for the four molecules studied. Anisotropy parameters are averaged over the energy region displaying long time dynamics (there was very little variation over this region). Error bars are one standard deviation. The rise in the anisotropy is fitted with a singular exponential, whose time scale τ_β is displayed on the charts. The vertical dashed line denotes the point where the time scale changes from linear to logarithmic.

4.2.4 Supporting Calculations

Calculations to aid the interpretation of the experimental results were undertaken by James Thompson and Martin Paterson, with some help from Therese Bergendahl. Ground state geometries of the four molecules phenol, catechol, resorcinol, and hydroquinone were optimised using density functional theory (B3LYP) in conjunction with the aug-cc-pVDZ basis set. Vertical excitation energies and oscillator strengths were calculated using equation of motion coupled cluster theory including single and double excitations (EOM-CCSD)[64] with the aug-cc-pVDZ basis; this is equivalent to linear response (LR) coupled cluster theory for excitation energies. The carbon and oxygen core 1s orbitals were frozen in the correlated calculations. The effect of perturbative triples on the excitation energies were examined for phenol and catechol using the CCR(3) method,[65] which gives a non-iterative perturbative correction to LR-CCSD excitation energies, such that excitation energies for singly-excited states are correct through third order in the fluctuation potential. It was found that triples affected the excitation by less than 0.15 eV and, in general, their inclusion had only a negligible effect. The vertical excitation energies and oscillator strengths are displayed in Table 4.3. In the case of resorcinol and hydroquinone which have multiple isomers, values are for isomer A and isomer B, respectively, as labelled in Figure 4.1. The oscillator strength of the optically bright ππ* transition in hydroquinone is almost twice that in the others. The πσ* state is dark in both phenol and hydroquinone but has a very small oscillator strength for catechol and resorcinol. Relaxed scans along the OH dissociation coordinate (valid until before the S_2/S_0 conical intersection) were performed using B3LYP/aug-cc-pVDZ, with those subsequent geometries used to obtain coupled cluster

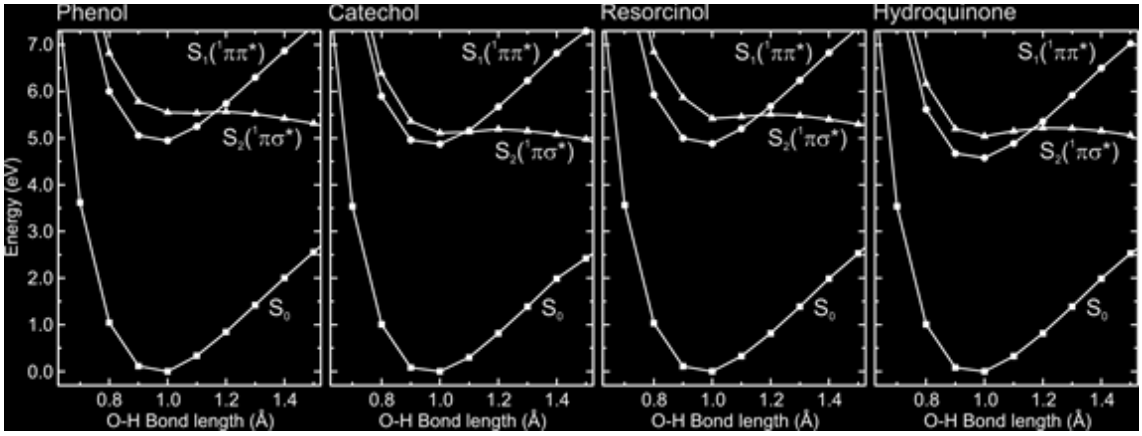


Figure 4.7: O-H stretching coordinate potential energy cuts obtained using EOM-CCSD/aug-cc-pVDZ, for phenol, catechol, resorcinol and hydroquinone. For more details see the main text.

ground (CCSD) and excited state (EOM-CCSD) energies. These procedures are very similar to the ones used for the calculations on indole and 5-hydroxyindole presented in Chapter 2 [66]. The results of these procedures are shown in Figure 4.7. Note that with catechol, the cut is taken along the non-hydrogen bonded O-H coordinate (see Figure 4.8). The resorcinol and hydroquinone cuts are, once again, for isomer A and isomer B respectively and in the case of resorcinol, the dissociation is along the H atom that is directed away from the other OH group. In all cases our calculations place the bottom of the S_1 state potential-well ~0.65 eV too high in energy relative to the position that would be predicted based on the known S_1 origins and O-H zero-point energies. However, such discrepancies are common at the level of theory used in these types of system and our results are in good agreement with those previously reported for phenol using similar methods (as summarised in Table I of Ref. 34). With respect to the phenol S_1 vertical excitation energy shown in Table 4.3, the differences in the calculated S_1 energies for catechol, resorcinol and hydroquinone agree extremely well with the experimentally observed shifts in the positions of the various S_1 origins. Complete-active-space self-consistent-field (CASSCF) calculations were completed in regions of strong non-adiabatic coupling to generate qualitatively correct wavefunctions. Fully relaxed geometry optimisations were performed for the $^1\pi\pi^*$ minimum and the $^1\pi\pi^*/^1\pi\sigma^*$ conical intersection seam. Details of the branching space vectors that lift the degeneracy at the seam minimum are shown in Figure 4.8 for all four molecules. These

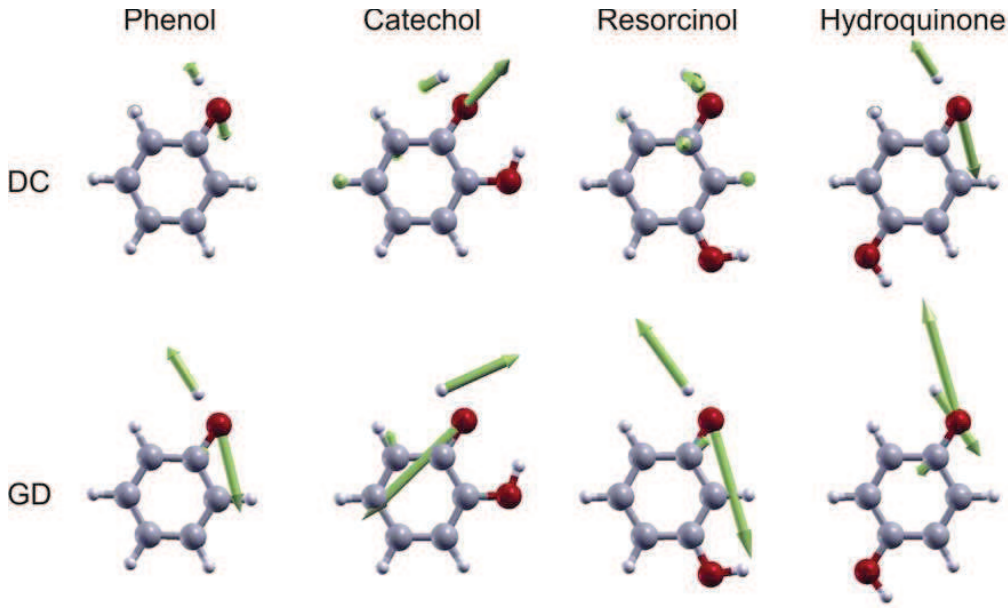


Figure 4.8: Branching space vectors for the $S_1(^1\pi\pi^*)/S_2$ ($^1\pi\sigma^*$) conical intersection as obtained from CASSCF calculations performed on all four of the molecules used in the present study. The derivative coupling vector (DC) and the gradient difference vectors (GD) define the directions in which the degeneracy is lifted when moving away from the conical intersection point.

are seen to involve motion mainly on the dissociating OH bond, with the states coupled through the OH stretch and some out-of-plane bending motion (the extent of which varies across the series). The Gaussian program [67] was used for the B3LYP, EOM-CCSD, and CASSCF calculations, while the Dalton program [68] was used for the LR-CCSD and CCR(3) calculations.

For the phenol molecule, a recent calculation at the CASPT2(10,10)/aug(O)-AVTZ level of theory [27] has proved extremely accurate in predicting the experimentally observed barrier height formed by the S_1/S_2 conical intersection along the O-H coordinate and also in modelling the experimentally determined H atom tunnelling rate along the O-H coordinate following excitation to the S_1 origin [14]. The tunnelling rate calculations were performed using a 1D semi-classical Brillouin-Kramers-Wentzel (BKW) approach [69]. Given that this method is extremely sensitive to small variations in barrier area, this is a remarkable result: for example, we find that changes in barrier area of only 3% influences the tunnelling rate by more than 30%. Our BKW calculations gave the following tunnelling rates: phenol 142 ps; catechol 111 fs; resorcinol 4.4 ns; and hydroquinone 303 ps. This is in very rough agreement with our measured values. These recent phenol calculations provide a strong indication of the accuracy of our excited state potentials generated for phenol. Comparing our EOM-CCSD calculations for phenol to the CASPT2 result, we find that our approach underestimates the S_1/S_2 conical intersection barrier height by ~0.2 eV. However, our data does produce a very similar O-H bond distance at which this conical intersection is located (~1.2 Å), and the shape of both the S_1 and S_2 potentials are qualitatively correct. We therefore take the potential cuts in Figure 4.7 as sufficiently reliable to enable a qualitative discussion of the change in barrier characteristics across the four molecules used in this present study. It is interesting to note from Figure 4.7 that the barrier area (with respect to the O-H zero-point energy) in catechol is approximately an order of magnitude smaller than in the other three molecules.

	Phenol		Catechol		Resorcinol		Hydroquinone	
	E / eV	*f*	*E* / eV	*f*	*E* / eV	*f*	*E* / eV	*f*
$S_1(^1\pi\pi^*)$	4.95	0.046	4.88	0.058	4.90	0.047	4.57	0.107
$S_2(^1\pi\sigma^*)$	5.55	0.000	5.17	0.001	5.41	0.001	5.12	0.000

Table 4.3: Calculated EOM-CCSD/aug-cc-pVDZ excitation energies and oscillator strengths. For additional details, see the main text.

4.3 Discussion

As has already been discussed in Section 4.1, many groups have looked to study the dynamics of phenol. Also, Ashfold's recent study has shed much new light on the dynamics of the catechol system [56], revealing that it shares many of the same dynamical characteristics as phenol. Although some subtle differences are seen in these two systems, the same basic relaxation routes appear to be in operation. To add weight to this, our calculations show in all four molecules that the same molecular states are energetically accessible. In addition, the experimental data of all four systems show the same common features: there are two exponentially decaying processes in all cases with the associated decay associated spectra overlapping the same spectral region. There are also similar photoelectron angular distributions with similar time-dependent evolutions. In the following discussion we therefore adopt the basic assumption that the relaxation pathways present in all four molecules are the same – although some clear differences also exist as will be discussed and explained in due course. On the basis of this assumption, we therefore frame our interpretation of these results in terms of the two different timescales observed, rather than considering the results of each molecule individually.

4.3.1 Long-time (>10 ps) Dynamics

There seems to be some consensus among the literature, as discussed in the introduction, that at excitation wavelengths longer than 248 nm one decay pathway for the initially prepared S_1 state in phenol is via tunnelling under the barrier formed by the S_1/S_2 conical intersection along the O-H stretching coordinate. At the 267 nm pump wavelength used in our present study, we are sitting ~2850 cm^{-1} below the (experimentally determined) S_1/S_2 conical intersection and are not able to access the S_2 state directly. We must reasonably attribute a percentage of the long-lived (τ_2 = 980 ps) time-constant in phenol to decay of the S_1 state via a tunnelling mechanism. This is due to the fact that structured H fast atom distributions have also been observed by Ashfold and co-workers in this energy region (see Table II of Ref. 28). For this process to occur, vibronically induced mixing of the S_1 and S_2 states is required and strong evidence for this interaction is evident in the highly anisotropic ($\beta_2 \sim 0.9$) photoelectron angular distributions we observe, even at very long pump-probe delay times. In the Franck-Condon region, the S_2 state is known to exhibit significant 3*s* Rydberg character [19].

As a simple, atomic-like approximation, a single photon ionisation of a 3*s* orbital should give rise to photoelectron partial waves of *p* character, peaking along the laser polarization axis. In fact this is what we observe, as illustrated in Figure 4.2 and Figure 4.5, suggesting that the S_1 state contains significant S_2 electronic character. Another factor that corroborates well with this assumption is that we know that the *s*-orbital character of the S_1 state can possess no initial alignment, and we observe no significant contribution to the photoelectron angular distributions from a β_4 component. It also explains why no decrease in β_2 is observed over time as a possible consequence of rotational dephasing in the initially prepared S_1 state. We in fact observe an *increase* in β_2 during the initial sub picosecond dynamical evolution, as evidenced in Figure 4.6 – we shall return to discuss this shortly.

Vibronic interactions between the S_1 and S_2 states are mediated by vibrational modes of a_2 symmetry. Since such modes are inaccessible by direct single-photon absorption, this suggests that some element of intramolecular vibrational redistribution (IVR) is required to promote this mixing and hints at the possible origin of the short time dynamics we see in our data, as is discussed further in Section 4.3.2. Our value of the overall S_1 lifetime following 267 nm excitation, τ_2 = 980 ps, lies between the values reported by Stavros and co-workers at 275 nm and 258 nm (1900 ps and 900 ps respectively) [14]. Although this work only required a single exponential decay to fit their observed parent ion transients data, these authors have also observed a bi-exponential decay in more recent, as yet unpublished data recorded at 267 nm [70]. This agrees with our own findings. Since our overall S_1 lifetime is shorter than their observed tunnelling lifetime (2.4 ns), another, competing pathway for long-time decay of the S_1 state is also possibly operative, with a rate that increases to shorter excitation wavelengths. However, given that our current experiments are "blind" to the population of lower lying, highly vibrationally excited electronic states (such as S_0) due to unfavourable Franck-Condon overlap with the cation, we are unable to comment on this further and it remains an open question.

If we now look at the catechol data, the most striking deviation from the phenol data is the much reduced S_1 lifetime (12.1 ps *vs*. 980 ps). This lifetime, however, is still sufficiently long-lived to suggest that at 267 nm we are not in a regime where significant direct excitation to the dissociative S_2 state (with subsequent prompt dissociation along the O-H stretching coordinate) is taking place. Ashfold and co-workers have suggested that the onset for such direct excitation may lie between 265-

270 nm. This is at a lower energy, relative to the S_1 origin, than in the case of phenol. They also postulate that a very similar H atom tunnelling mechanism to that present in phenol is operative at longer excitation wavelengths [56]. The calculations that we have carried out, presented in Figure 4.7, clearly show an order of magnitude reduction in the area under the barrier formed by the S_1/S_2 conical intersection along the O-H stretching coordinate in catechol as compared to phenol. On the foundation of this evidence it is reasonable to rationalize that this major decrease in the S_1 lifetime is due to an increase in the H atom tunnelling rate as a consequence of the suppression of the barrier. Similarly to phenol, this process appears to be mediated by a vibronic interaction that mixes the electronic character of the two states, as is verified by the time-dependent progression of the anisotropy in the photoelectron angular distributions for catechol. Our catechol S_1 lifetime (12.1 ps) is also in good agreement with Stavros' as yet unpublished study: at 267 nm excitation, they observe lifetimes of 11.5 ps and 10.5 ps for the overall S_1 lifetime (obtained from a transient ion-yield measurement) and fast H atom appearance time (characteristic of dissociation on the S_2 surface), respectively [70].

Also, and perhaps more significantly, this group have also observed the fast H atom appearance time to be largely unchanging over the 250-280.5 nm excitation range. This appears to suggest that H-atom tunnelling proceeds entirely from the zero-point energy of the S_1 state along the O-H coordinate. Such an assertion clearly places the S_1/S_2 barrier at considerably higher excitation energy (comparable to phenol), which appears to be inconsistent with the relatively short lifetimes observed. The S_1 state of catechol is known to be non-planar, in contrast to the other three molecules investigated in this present study, with the non-hydrogen bonded OH group pointing out of the plane defined by the rest of the molecular framework [41]. The role of OH torsional excitation within the S_1 state has also been implemented in mediating the catechoxy radical product state distribution following dissociation [56]. We propose, therefore, that interpreting experimental data from catechol using a simple 1D model directed along the O-H stretching coordinate may be inadequate to fully explain the excited state relaxation dynamics, and more extensive calculations will ultimately be required.

We now turn our attention to the final two systems under study, resorcinol and hydroquinone. Our calculations predict the height and area under the S_1/S_2 barrier in these two molecules to be very similar to that predicted in phenol. The τ_2 lifetimes we obtained for resorcinol and hydroquinone are quite different however, being > 1 ns and

440 ps, respectively. Compared to the corresponding value for phenol (980 ps), the hydroquinone lifetime seems to be in reasonable qualitative agreement with what might be expected on the basis of our calculations. This is because H atom tunnelling may occur along either of the two O-H coordinates present in this system, resulting in an apparent increase in decay rate by a factor of 2. This is obviously based on the assumption that H atom tunnelling is a major relaxation pathway in this system but, given that in phenol and catechol such a mechanism is clearly present, this would seem to be a good assumption. This assumption is also supported by features that are consistent across all of our experimental data, such as the photoelectron angular distribution anisotropy and its associated temporal evolution. Having no information in this experiment of how the lifetime evolves with excitation wavelength, we are unable to comment further on any other possible competing processes, such as the previously mentioned internal conversion to a vibrationally excited ground state driven by spectator modes orthogonal to the O-H coordinate that has been postulated in phenol. This is an issue that can be addressed in future studies, as expanded upon in Sections 5.3.2 and 5.3.3. The tunnelling assumption would also appear to be reasonable for the resorcinol data, for similar reasons as those stated above. Given that a factor of 2 increase in tunnelling rate might also be anticipated for this system, the extremely long τ_2 value is difficult to reconcile on the basis of our supporting calculations. However, as discussed earlier, the highly-sensitive exponential dependence of tunnelling rates on barrier characteristics means that even relatively small changes in the energetic positions and/or shape of the S_1 and S_2 potentials may have a profound effect on lifetime. In this instance our calculations may lack sufficient accuracy to fully address these subtleties.

4.3.2 Short-time (<1 ps) Dynamics

Since we have assigned and understood the long time dynamics, we now look at the shorter time dynamics. The values of τ_1 range between 320 and 990 fs, these are clearly too long to be attributable to “ballistic” motion of an initially prepared vibrational wavepacket out of the Franck-Condon window, resulting in a decrease in ionization efficiency. As discussed previously, we may also rule out excitation to superexcited states, due to the fact that our observed photoelectron energy distributions do not show signal below the ionisation potential. An alternative possibility is that these fast processes result from rapid non-adiabatic decay to lower lying electronic states. Time-

resolved photoelectron studies of the S_1 state in benzene reported by Fielding and co-workers have observed sub-picosecond dynamical timescales following UV excitation in the "channel 3 region" [71, 72]. This was attributed to two competing relaxation pathways: ultrafast intersystem crossing to the T_2 triplet state (with subsequent relaxation *via* T_1), and internal conversion directly to S_0. Theoretical studies suggest that for both of these routes the relevant conical intersections lie along a reaction coordinate where benzene evolves towards prefulvene, which is a non-planar, biradical structure [73, 74]. A similar prediction has also been made for phenol by Domcke and coworkers [75], although the energetic barrier to access the prefulvenic decay channel is calculated to be 6370 cm^{-1}, which is more than 2000 cm^{-1} above the S_1/S_2 conical intersection. However, in the context of this present work, the involvement of pre-fulvenic types of structure rapidly driving population to new electronic states is difficult to reconcile with our observations; the timescale for such a rearrangement of the molecular framework should be highly sensitive to the number and position of the substituent groups attached to the benzene ring. This is not what we observe, as the τ_1 values seen to be broadly similar in all cases. Phenol, resorcinol and hydroquinone have almost identical lifetimes of 720 fs, 990 fs and 880 fs, respectively. Within the "view" of the reaction coordinate(s) afforded by our chosen pump wavelengths, we suggest that we see no strong evidence to support the involvement of sub-picosecond internal conversion or inter-system crossing pathways to lower lying states. We note, however, that we cannot rule out such a possibility definitively as we are effectively blind to their direct population. The vacuum ultraviolet setup described in Section 5.3.3 would be an ideal probe source that would allow us to see further along the reaction coordinate and detect these lower lying states.

One alternative possibility for the origin of τ_1 in our data is ultrafast intramolecular vibrational redistribution (IVR) on the S_1 potential surface. This could lead to a modified Franck-Condon overlap with the cation and an associated decrease (or, in the case of resorcinol, increase) in ionization signal (see Section 1.2.2 for a discussion of the Franck-Condon principle). Time-resolved IR-UV pump-probe studies by Abel and co-workers, initially populating the CH-stretch overtone or the CH-stretch-CC-stretch combination in the S_0 state of benzene and the difluorobenzenes, have reported a two stage IVR process with an initial redistribution of vibrational energy into "doorway states" taking place on a sub-picosecond timescale. Subsequent "statistical" or "distributive" IVR was then also observed at longer pump-probe delays [76]. This work

also demonstrated that the IVR rate increased in systems possessing lower symmetry, which is a commonly encountered phenomenon due to less restrictive coupling requirements leading to an effective increase in the density of states. Ultrafast IVR has also been suggested to account for sub-ps dynamical observations on the S_1 potential surface of chlorobenzene [77]. We therefore suggest that some form of IVR mechanism, mediated by vibronic rather than anharmonic coupling, is the most likely origin of the τ_1 decay seen in our data. For the four molecules investigated in this present study, we are sitting ~1100-3950 cm^{-1} above the S_1 origin, which should be well above the threshold for the onset of such a process in all cases. We note that in hydroquinone this threshold has been experimentally determined to be 1650 cm^{-1} [52], and that in the S_1 state of anisole (the methoxy-substituted analogue of phenol) it has been reported at 940 cm^{-1} [78].

An IVR interpretation is also supported by the previously discussed requirement that some form of vibronic interaction must be required to promote the S_1/S_2 mixing that is apparent in the long-lived anisotropy of the PADs we observe – particularly since this mixing is mediated by vibrational modes of a_2 symmetry that are not optically accessible in the initial one-photon excitation [27]. The role of IVR also accounts for the shorter τ_1 lifetime (320 fs) seen in catechol relative to the other systems under study, as the non-planar S_1 state possesses a lower symmetry than phenol, resorcinol and hydroquinone. Additionally, an IVR mechanism provides a simple explanation for the rising signals seen in resorcinol (and the associated negative amplitude features present in the τ_1 DAS): in contrast to phenol, catechol and hydroquinone, where vibrational redistribution reduces the Franck-Condon overlap with the cation, in resorcinol it may lead to an increase in some regions of the photoelectron spectrum. Such variations in the extent to which IVR changes the Franck-Condon overlap in photoionization may also explain the mono-exponential decay observed in the phenol ion-yield signal reported by Stavros and co-workers at 253 nm [14].

In phenol and resorcinol, the IVR interpretation also sits well with the observation that τ_1 is well matched to τ_β: coupling into modes that enhance S_1/S_2 mixing (as described by τ_1) leads to an evolution of the S_1 electronic character (as described by τ_β). In catechol ($\tau_1 < \tau_\beta$) and hydroquinone ($\tau_1 > \tau_\beta$) the situation is more complex and we suggest that the differences between τ_1 and τ_β may be attributable to only a subset of the modes populated during the IVR process having an influence on the S_1/S_2 interaction.

For the case of all four systems under study we therefore attribute the τ_1 lifetime to changes in the *vibrational character* within the S_1 state and the τ_β lifetime to the evolution of the *electronic character* of the S_1 state. We note that similar assertions have previously been put forward in excited state studies of benzene and toluene made by Suzuki and co-workers [79]. The fact that different timescales may be observed for these two processes is a powerful illustration of the highly differential nature of the time-resolved imaging approach. Finally, we suggest that we are unable to observe any subsequent second-tier "statistical" or "distributive" IVR processes in our data due to the relatively large number of vibrational states initially populated by the broadband pump pulse used in our experiments. In effect, the initial IVR step is therefore essentially "quasi-statistical" in nature and secondary vibrational energy redistribution does not produce any appreciable further changes in the photoionization cross-section at longer pump-probe delays. Hence our data may be accurately modelled using just two exponential time constants.

4.4 Conclusions

The S_1 ($\pi\pi^*$) relaxation dynamics following 267 nm excitation in the commonly occurring biological motifs phenol, catechol, resorcinol and hydroquinone were investigated using time-resolved photoelectron imaging. In all four cases we attribute the rapid (sub picosecond) dynamics to intramolecular vibrational redistribution on the S_1 potential surface. This process enhances an interaction between the S_1 state and the nearby S_2 state, which is of 3s Rydberg character in the vertical Franck-Condon region and evolves towards $\pi\sigma^*$ character at extended O-H bond distances. The mixing of this Rydberg character into the S_1 state is readily apparent in the temporal evolution of the highly anisotropic photoelectron angular distributions (positive β_2) that we observe. We suggest that, more generally, the observation of such anisotropy may provide a strong, direct signature of $\pi\sigma^*$ state participation in the dynamics. These states are important in the dynamics of many systems containing OH, NH and SH groups; this signature anisotropy has been observed before, as has been recently illustrated in aniline by Fielding and co-workers [80]. This could be particularly useful in systems where $\pi\sigma^*$ participation cannot always be unambiguously determined. Less differential techniques, like our own work on indole and 5-hydroxyindole discussed in Chapter 2 carried out using a magnetic bottle photoelectron spectrometer [66], are unable to detect this anisotropy. This shows the advantage of photoelectron imaging techneques. We note,

however, that such dynamical signatures may be easily obscured or misinterpreted if multi-photon ionisation schemes (potentially proceeding via higher-lying Rydberg states) are employed. Section 5.3.3 discusses other advantages of using one photon ionisation schemes over two photon schemes. The longer time dynamics we observe, characterised by τ_2, occur on a strikingly different timescale in catechol when compared to the other three molecules under study. On the basis of our supporting calculations which show the barrier formed by the S_1/S_2 conical intersection along the O-H stretching coordinate to be much lower in catechol then in the other three molecules, we attribute this primarily to differences in the H atom tunnelling rate under the barrier formed by the S_1/S_2 conical intersection.

Great care must be taken when drawing detailed mechanistic conclusions in complex systems on the basis of studies performed at just a single absorption wavelength. However, we believe there is a sufficient body of previous work on these molecules in the literature to allow us to present our interpretations of the data with confidence. Additional experiments with different pump wavelengths were not possible because of (a) limitations enforced by our current laser set-up, restricting the use of fully tuneable UV output for both the probe and pump pulses (the remedy for this is discussed in Section 5.3) (b) our desire, where possible, to perform 'clean' experiments where the short-time dynamics are not potentially obscured by probe-pump dynamics evolving in the negative time delay direction, and (c) our wish to avoid using multi-photon ionisation schemes in the probe step, as this can potentially induce transitions to additional, higher lying electronically excited states. These schemes may obscure the dynamics of interest, and may give rise to different photoelectron angular distributions.

The results presented here are submitted for publication, and provide a very satisfactory conclusion to my studies. The photoelectron spectrometer and the software to collect and analyse data have proved their ability to collect high quality data, and analyse it in an appropriate manner, and should be useful for researching many more systems in the future.

4.5 References

[1] A. Sur and P. M. Johnson, *Radiationless Transitions in Gas-Phase Phenol and the Effects of Hydrogen-Bonding*, J. Chem. Phys., **84**, 1206, (1986).

[2] R. J. Lipert, G. Bermudez, and S. D. Colson, *Pathways of S_1 Decay in Phenol, Indoles, and Water Complexes of Phenol and Indole in a Free Jet Expansion*, J. Phys. Chem., **92**, 3801, (1988).

[3] M. Takayanagi and I. Hanazaki, *Stimulated-Emission-Pumping Laser-Induced-Fluorescence Spectroscopy of Phenol and Anisole*, Las. Chem., **14**, 103, (1994).

[4] G. Berden, W. L. Meerts, M. Schmitt, and K. Kleinermanns, *High Resolution UV Spectroscopy of Phenol and the Hydrogen Bonded Phenol-Water Cluster*, J. Chem. Phys., **104**, 972, (1996).

[5] W. Roth, P. Imhof, M. Gerhards, S. Schumm, and K. Kleinermanns, *Reassignment of Ground and First Excited State Vibrations in Phenol*, Chem. Phys., **252**, 247, (2000).

[6] T. Ebata, M. Kayano, S. Sato, and N. Mikami, *Picosecond IR-UV Pump-Probe Spectroscopy. IVR of OH Stretching Vibration of Phenol and Phenol Dimer*, J. Phys. Chem. A, **105**, 8623, (2001).

[7] C. Ratzer, J. Kupper, D. Spangenberg, and M. Schmitt, *The Structure of Phenol in the S_1 State Determined by High Resolution UV-Spectroscopy*, Chem. Phys., **283**, 153, (2002).

[8] C. M. Tseng, Y. M. Choi, C. L. Huang, C. K. Ni, Y. T. Lee, and M. C. Lin, *Photodissociation of Nitrosobenzene and Decomposition of Phenyl Radical*, J. Phys. Chem. A, **108**, 7928, (2004).

[9] Y. Yamada, N. Mikami, and T. Ebata, *Real-Time Detection of Doorway States in the Intramolecular Vibrational Energy Redistribution of the OH/OD Stretch Vibration of Phenol*, J. Chem. Phys., **121**, 11530, (2004).

[10] C. M. Tseng, Y. T. Lee, M. F. Lin, C. K. Ni, S. Y. Liu, Y. P. Lee, Z. F. Xu, and M. C. Lin, *Photodissociation Dynamics of Phenol*, J. Phys. Chem. A, **111**, 9463, (2007).

[11] M. L. Hause, Y. H. Yoon, A. S. Case, and F. F. Crim, *Dynamics at Conical Intersections: The Influence of O-H Stretching Vibrations on the Photodissociation of Phenol*, J. Chem. Phys., **128**, 104307, (2008).

[12] A. Iqbal, L. J. Pegg, and V. G. Stavros, *Direct Versus Indirect H Atom Elimination from Photoexcited Phenol Molecules*, J. Phys. Chem. A, **112**, 9531, (2008).

[13] A. Iqbal, M. S. Y. Cheung, M. G. D. Nix, and V. G. Stavros, *Exploring the Time-Scales of H-Atom Detachment from Photoexcited Phenol-h(6) and Phenol-d(5): Statistical vs Nonstatistical Decay*, J. Phys. Chem. A, **113**, 8157, (2009).

[14] G. M. Roberts, A. S. Chatterley, J. D. Young, and V. G. Stavros, *Direct Observation of Hydrogen Tunneling Dynamics in Photoexcited Phenol*, J. Phys. Chem. Lett., **3**, 348, (2012).

[15] M. Krauss, J. O. Jensen, and H. F. Hameka, *Electronic-Structure of the Excited-States and Phenol Fluorescence*, J. Phys. Chem., **98**, 9955, (1994).

[16] J. Lorentzon, P. A. Malmqvist, M. Fulscher, and B. O. Roos, *A CASPT2 Study of the Valence and Lowest Rydberg Electronic States of Benzene and Phenol*, Theor. Chim. Acta., **91**, 91, (1995).

[17] S. Schumm, M. Gerhards, and K. Kleinermanns, *Franck-Condon Simulation of the $S_1 \rightarrow S_0$ Spectrum of Phenol*, J. Phys. Chem. A, **104**, 10648, (2000).

[18] A. L. Sobolewski and W. Domcke, *Photoinduced Electron and Proton Transfer in Phenol and Its Clusters with Water and Ammonia*, J. Phys. Chem. A, **105**, 9275, (2001).

[19] A. L. Sobolewski, W. Domcke, C. Dedonder-Lardeux, and C. Jouvet, *Excited-State Hydrogen Detachment and Hydrogen Transfer Driven by Repulsive $^1\pi\sigma^*$ States: A New Paradigm for Nonradiative Decay in Aromatic Biomolecules*, Phys. Chem. Chem. Phys., **4**, 1093, (2002).

[20] M. Abe, Y. Ohtsuki, Y. Fujimura, Z. G. Lan, and W. Domcke, *Geometric Phase Effects in the Coherent Control of the Branching Ratio of Photodissociation Products of Phenol*, J. Chem. Phys., **124**, 224316, (2006).

[21] M. de Groot and W. J. Buma, *A Time-Dependent Density Functional Study of Vibrationally Resolved Excitation, Emission, and Ionization Spectra of the S_1 State of Phenol*, Chem. Phys. Lett., **420**, 459, (2006).

[22] M. Miura, Y. Aoki, and B. Champagne, *Assessment of Time-Dependent Density Functional Schemes for Computing the Oscillator Strengths of Benzene, Phenol, Aniline, and Fluorobenzene*, J. Chem. Phys., **127**, 084103, (2007).

[23] O. P. J. Vieuxmaire, Z. Lan, A. L. Sobolewski, and W. Domcke, *Ab Initio Characterization of the Conical Intersections Involved in the Photochemistry of Phenol*, J. Chem. Phys., **129**, 224307, (2008).

[24] Z. G. Lan, W. Domcke, V. Vallet, A. L. Sobolewski, and S. Mahapatra, *Time-Dependent Quantum Wave-Packet Description of the $^1\pi\sigma^*$ Photochemistry of Phenol*, J. Chem. Phys., **122**, 224315, (2005).

[25] H. An and K. K. Baeck, *Quantum Wave Packet Propagation Study of the Photochemistry of Phenol: Isotope Effects (Ph-OD) and the Direct Excitation to the $^1\pi\sigma^*$ State*, J. Phys. Chem. A, **115**, 13309, (2011).

[26] M. N. R. Ashfold, G. A. King, D. Murdock, M. G. D. Nix, T. A. A. Oliver, and A. G. Sage, *$\pi\sigma^*$ Excited States in Molecular Photochemistry*, Phys. Chem. Chem. Phys., **12**, 1218, (2010).

[27] R. N. Dixon, T. A. A. Oliver, and M. N. R. Ashfold, *Tunnelling Under a Conical Intersection: Application to the Product Vibrational State Distributions in the UV Photodissociation of Phenols*, J. Chem. Phys., **134**, 194303, (2011).

[28] G. A. King, T. A. A. Oliver, M. G. D. Nix, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy Studies of the Ultraviolet Photolysis of Phenol-d(5)*, J. Phys. Chem. A, **113**, 7984, (2009).

[29] M. G. D. Nix, A. L. Devine, R. N. Dixon, and M. N. R. Ashfold, *Observation of Geometric Phase Effect Induced Photodissociation Dynamics in Phenol*, Chem. Phys. Lett., **463**, 305, (2008).

[30] M. N. R. Ashfold, A. L. Devine, R. N. Dixon, G. A. King, M. G. D. Nix, and T. A. A. Oliver, *Exploring Nuclear Motion through Conical Intersections in the UV Photodissociation of Phenols and Thiophenol*, P. Natl. Acad. Sci. USA, **105**, 12701, (2008).

[31] G. A. King, A. L. Devine, M. G. D. Nix, D. E. Kelly, and M. N. R. Ashfold, *Near-UV Photolysis of Substituted Phenols*, Phys. Chem. Chem. Phys., **10**, 6417, (2008).

[32] A. L. Devine, M. G. D. Nix, B. Cronin, and M. N. R. Ashfold, *Near-UV Photolysis of Substituted Phenols, I: 4-fluoro-, 4-chloro- and 4-bromophenol*, Phys. Chem. Chem. Phys., **9**, 3749, (2007).

[33] M. G. D. Nix, A. L. Devine, B. Cronin, R. N. Dixon, and M. N. R. Ashfold, *High Resolution Photofragment Translational Spectroscopy Studies of the near Ultraviolet Photolysis of Phenol*, J. Chem. Phys., **125**, 133318, (2006).

[34] M. N. R. Ashfold, B. Cronin, A. L. Devine, R. N. Dixon, and M. G. D. Nix, *The Role of πσ* Excited States in the Photodissociation of Heteroaromatic Molecules*, Science, **312**, 1637, (2006).

[35] G. Pino, A. Oldani, E. Marceca, M. Fujii, S. I. Ishiuchi, M. Miyazaki, M. Broquier, C. Dedonder, and C. Jouvet, *Excited State Hydrogen Transfer Dynamics in Substituted Phenols and Their Complexes with Ammonia: ππ*-πσ* Energy Gap Propensity and Ortho-Substitution Effect*, J. Chem. Phys., **133**, 124313, (2010).

[36] C. P. Schick and P. M. Weber, *Ultrafast Dynamics in Superexcited States of Phenol*, J. Phys. Chem. A, **105**, 3725, (2001).

[37] C. A. Beck, *Near Ultraviolet Absorption Spectrum of Hydroquinone, Resorcinol, and Catechol*, J. Chem. Phys., **18**, 1135, (1950).

[38] T. M. Dunn, R. Tembreull, and D. M. Lubman, *Free-Jet Spectra and Structure of o-, m- and p-dihydroxybenzenes*, Chem. Phys. Lett., **121**, 453, (1985).

[39] T. Bürgi and S. Leutwyler, *O–H Torsional Vibrations in the S and S States of Catechol*, J. Chem. Phys., **101**, 8418, (1994).

[40] M. Gerhards, W. Perl, S. Schumm, U. Henrichs, C. Jacoby, and K. Kleinermanns, *Structure and Vibrations of Catechol and Catechol.$H_2O(D_2O)$ in the S_0 and S_1 State*, J. Chem. Phys., **104**, 9362, (1996).

[41] M. Gerhards, S. Schumm, C. Unterberg, and K. Kleinermanns, *Structure and Vibrations of Catechol in the S_1 State and Ionic Ground State*, Chem. Phys. Lett., **294**, 65, (1998).

[42] W. R. Duncan and O. V. Prezhdo, *Electronic Structure and Spectra of Catechol and Alizarin in the Gas Phase and Attached to Titanium*, J. Phys. Chem. B, **109**, 365, (2005).

[43] M. Gerhards, W. Perl, and K. Kleinermanns, *Rotamers and Vibrations of Resorcinol Obtained by Spectral Hole Burning*, Chem. Phys. Lett., **240**, 506, (1995).

[44] M. Gerhards, M. Schiwek, C. Unterberg, and K. Kleinermanns, *OH Stretching Vibrations in Aromatic Cations: IR/PIRI Spectroscopy*, Chem. Phys. Lett., **297**, 515, (1998).

[45] M. Gerhards, C. Unterberg, and S. Schumm, *Structure and Vibrations of Dihydroxybenzene Cations and Ionization Potentials of Dihydroxybenzenes Studied by Mass Analyzed Threshold Ionization and Infrared Photoinduced Rydberg Ionization Spectroscopy as Well as Ab Initio Theory*, J. Chem. Phys., **111**, 7966, (1999).

[46] G. Myszkiewicz, W. L. Meerts, C. Ratzer, and M. Schmitt, *Structure Determination of Resorcinol Rotamers by High-Resolution UV Spectroscopy*, ChemPhysChem, **6**, 2129, (2005).

[47] A. Oikawa, H. Abe, N. Mikami, and M. Ito, *Electronic Spectra and Ionization Potentials of Rotational Isomers of Several Disubstituted Benzenes*, Chem. Phys. Lett., **116**, 50, (1985).

[48] R. Tembreull, T. M. Dunn, and D. M. Lubman, *Excited State Spectroscopy of Para Di-Substituted Benzenes in a Supersonic Beam Using Resonant Two Photon Ionization*, Spectrochim. Acta, Part A, **42**, 899, (1986).

[49] S. J. Humphrey and D. W. Pratt, *High Resolution $S_1 \leftarrow S_0$ Fluorescence Excitation Spectra of Hydroquinone. Distinguishing the cis and trans Rotamers by Their Nuclear Spin Statistical Weights*, J. Chem. Phys., **99**, 5078, (1993).

[50] W. Tzeng, K. Narayanan, C. Hsieh, and C. Tung, *A Study of the Excited State Structure and Vibrations of Hydroquinone by Ab Initio Calculations and Resonant Two-Photon Ionization Spectroscopy*, Spectrochim. Acta, Part A, **53**, 2595, (1997).

[51] G. Patwari, S. Doraiswamy, and S. Wategaonkar, *Hole-Burning Spectroscopy of Jet-Cooled Hydroquinone*, Chem. Phys. Lett., **289**, 8, (1998).

[52] G. Patwari, S. Doraiswamy, and S. Wategaonkar, *Spectroscopy and IVR in the S_1 State of Jet-Cooled p-Alkoxyphenols*, J. Phys. Chem. A, **104**, 8466, (2000).

[53] M. Onda, K. Hasunuma, T. Hashimoto, and I. Yamaguchi, *Microwave Spectrum of Catechol (1,2-Dihydroxybenzene)*, J. Mol. Struct., **159**, 243, (1987).

[54] W. Caminati, S. Di Bernardo, L. Schäfer, S. Q. Kulp-Newton, and K. Siam, *Investigation of the Molecular Structure of Catechol by Combined Microwave Spectroscopy and Ab Initio Calculations*, J. Mol. Struct., **240**, 263, (1990).

[55] C. Puebla and T. K. Ha, *A Theoretical Study of Conformations and Rotational Barriers in Dihydroxybenzenes*, J. Mol. Struct. THEOCHEM, **204**, 337, (1990).

[56] G. A. King, T. A. A. Oliver, R. N. Dixon, and M. N. R. Ashfold, *Vibrational Energy Redistribution in Catechol During Ultraviolet Photolysis*, Phys. Chem. Chem. Phys., **14**, 3338, (2012).

[57] R. A. Livingstone, J. O. F. Thomson, M. Iljina, R. J. Donaldson, B. J. Sussman, and D. Townsend, *Time-Resolved Photoelectron Imaging of Excited State Relaxation Dynamics in Phenol, Catechol, Resorcinol and Hydroquinone*, J. Chem. Phys., **Accepted**, (2012).

[58] U. Even, J. Jortner, D. Noy, N. Lavie, and C. Cossart-Magos, *Cooling of Large Molecules Below 1 K and He Clusters Formation*, J. Chem. Phys., **112**, 8068, (2000).

[59] NIST_Webbook, *http://webbook.nist.gov/chemistry*, (2012).

[60] R. J. Lipert and S. D. Colson, *Accurate Ionization Potentials of Phenol and Phenol-(HO) from the Electric Field Dependence of the Pump–Probe Photoionization Threshold*, J. Chem. Phys., **92**, 3240, (1990).

[61] M. Gerhards, C. Unterberg, and S. Schumm, *Structure and Vibrations of Dihydroxybenzene Cations and Ionization Potentials of Dihydroxybenzenes Studied by Mass Analyzed Threshold Ionization and Infrared Photoinduced Rydberg Ionization Spectroscopy as Well as Ab Initio Theory*, J. Chem. Phys., **111**, 7966, (1999).

[62] T. Kobayashi and S. Nagakura, *Photoelectron Spectra of Substituted Benzenes*, B. Chem. Soc. Jpn., **47**, 2563, (1974).

[63] M. H. Palmer, W. Moyes, M. Speirs, and J. N. A. Ridyard, *The Electronic Structure of Substituted Benzenes; Ab Initio Calculations and Photoelectron Spectra for Phenol, the Methyl-and Fluoro-Derivatives, and the Dihydroxybenzenes*, J. Mol. Struct., **52**, 293, (1979).

[64] J. F. Stanton and R. J. Bartlett, *The Equation of Motion Coupled-Cluster Method - a Systematic Biorthogonal Approach to Molecular-Excitation Energies, Transition-Probabilities, and Excited-State Properties*, J. Chem. Phys., **98**, 7029, (1993).

[65] O. Christiansen, H. Koch, and P. Jørgensen, *Perturbative Triple Excitation Corrections to Coupled Cluster Singles and Doubles Excitation Energies*, J. Chem. Phys., **105**, 1451, (1996).

[66] R. Livingstone, O. Schalk, A. E. Boguslavskiy, G. R. Wu, L. T. Bergendahl, A. Stolow, M. J. Paterson, and D. Townsend, *Following the Excited State Relaxation Dynamics of Indole and 5-Hydroxyindole Using Time-Resolved Photoelectron Spectroscopy*, J. Chem. Phys., **135**, 194307, (2011).

[67] *Gaussian 09, Revision A.2, M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, Ö. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, and D. J. Fox, Gaussian, Inc., Wallingford CT, 2009*

[68] *DALTON, a molecular electronic structure program, Release 2.0 (2005), see http://www.kjemi.uio.no/software/dalton/dalton.html 2005.*

[69] R. J. Le Roy and W. K. Liu, *Energies and Widths of Quasibound Levels (Orbiting Resonances) for Spherical Potentials*, J. Chem. Phys., **69**, 3622, (1978).

[70] G. M. Roberts and V. G. Stavros, Personal Communication.

[71] D. Parker, R. Minns, T. Penfold, G. Worth, and H. Fielding, *Ultrafast Dynamics of the S_1 Excited State of Benzene*, Chem. Phys. Lett., **469**, 43, (2009).

[72] R. Minns, D. Parker, T. Penfold, G. Worth, and H. Fielding, *Competing Ultrafast Intersystem Crossing and Internal Conversion in the "Channel 3" Region of Benzene*, Phys. Chem. Chem. Phys., **12**, 15607, (2010).

[73] T. Penfold and G. Worth, *The Effect of Molecular Distortions on Spin-Orbit Coupling in Simple Hydrocarbons*, Chem. Phys., **375**, 58, (2010).

[74] Q. Li, D. Mendive-Tapia, M. J. Paterson, A. Migani, M. J. Bearpark, M. A. Robb, and L. Blancafort, *A Global Picture of the S_1/S_0 Conical Intersection Seam of Benzene*, Chem. Phys., **377**, 60, (2010).

[75] O. P. J. Vieuxmaire, Z. Lan, A. L. Sobolewski, and W. Domcke, *S&D phenol*, J. Chem. Phys., **129**, 224307, (2008).

[76] R. S. von Benten, Y. Liu, and B. Abel, *Dynamical Consequences of Symmetry Breaking in Benzene and Difluorobenzene*, J. Chem. Phys., **133**, 134306, (2010).

[77] Y. Z. Liu, C. C. Qin, S. Zhang, Y. M. Wang, and B. Zhang, *Ultrafast Dynamics of the First Excited State of Chlorobenzene*, Acta Phys.-Chim. Sin., **27**, 965, (2011).

[78] R. Matsumoto, K. Sakeda, Y. Matsushita, T. Suzuki, and T. Ichimura, *Spectroscopy and Relaxation Dynamics of Photoexcited Anisole and Anisole-D3 Molecules in a Supersonic Jet*, J. Mol. Struct., **735**, 153, (2005).

[79] Y. I. Suzuki, T. Horio, T. Fuji, and T. Suzuki, *Time-Resolved Photoelectron Imaging of $S_2 \rightarrow S_1$ Internal Conversion in Benzene and Toluene*, J. Chem. Phys., **134**, 184313, (2011).

[80] R. Spesyvtsev, O. M. Kirkby, M. Vacher, and H. H. Fielding, *Shedding New Light on the Role of the Rydberg State in the Photochemistry of Aniline*, Phys. Chem. Chem. Phys., **14**, 9942, (2012).

5. Overview and Future Direction

This chapter draws the different sections of the thesis together, and looks forwards to how the research and equipment discussed here could be developed in the future. The main scope of my project has been to develop equipment that can be used for many years. This was an extensive body of work as is discussed in Chapter 3 and has been successfully used to conduct its first experiment, as detailed in Chapter 4. This section discusses some of the experiments that may be conducted with the new setup, and some of the enhancing extensions and technical upgrades to the equipment that could be made.

5.1. Model Biological Systems

Building on the experiences I gained through my time in Canada (Chapter 2), the experimental equipment and software was designed and set up (as described in Chapter 3). It is now fully operational and has the ability to study many model biological systems such as phenol and its derivatives, as discussed in Chapter 4. This section explores some of the interesting systems that, given more time, would have been good to investigate using this apparatus, and that the group here in Heriot-Watt University may investigate after I have left.

5.1.1. 5,6-Dihydroxyindole and Indole Derivatives

The purpose of the experiments reported in Chapters 2 and 4 is to work towards a better understanding of the 5,6-dihydroxyindole molecule that is a component of eumelanins. 5,6-dihydroxyindole (Figure 5.1a) decomposes rapidly in air, so is a difficult molecule to work with. In order to better understand the dynamics of this molecule it would be advantageous to look at the time-resolved photoelectron spectroscopy of 4-hydroxyindole, 5-hydroxyindole, 6-hydroxyindole, and 5,6-dihydroxyindole (see Figure 5.1). This would also extend the study conducted in chapter 4, investigating how the relative position of a hydroxyl group on an aromatic ring affects the dynamics of the molecule.

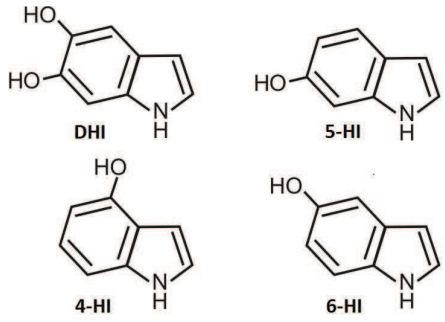


Figure 5.1: 5,6-dihydroxyindole and the indole derivatives 5-hydroxyindole, 4-hydroxyindole and 6-hydroxyindole.

The 5,6-dihydroxyindole molecule has been synthesised by Magnus Bebbington in Heriot-Watt. University. As preliminary work, the spectra in cyclohexane of these four molecules have recently been collected by James Thompson and are displayed in Figure 5.2. Encouragingly, 5,6-dihydroxyindole shows no change in the absorption spectrum or nuclear magnetic resonance spectrum after being heated for 2 days at 80^{o} C under helium. This suggests that molecular beam studies should be feasible without the molecule breaking down. Synthetic and physical chemistry groups collaborations should be exploited more to allow studies of molecules not easily commercially available.

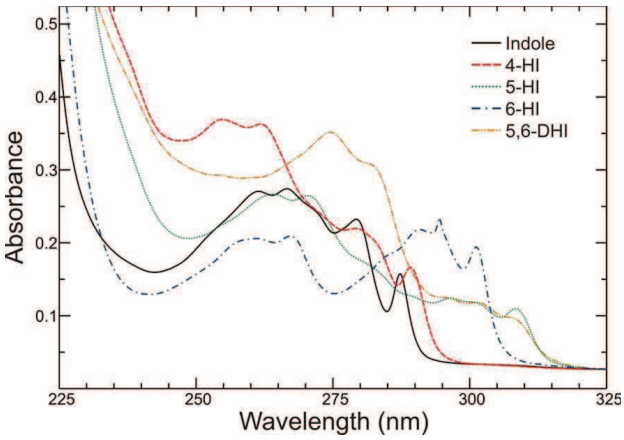


Figure 5.2: Recent unpublished absorption spectra of the four molecules in cyclohexane shown in Figure 5.1.

There is much current research on these important biological systems [1-22], as is summarised below. A femtosecond pump-probe degenerate transient absorption study by Simon and co-workers looked at the dynamics of the entire eumelanin system [16]. It showed a nonradiative process almost exclusively repopulating the ground state after excitation in the 320 – 400 nm range. A very rapid (<50 fs) decay was observed, although the ground state bleach recovery was seen to be a somewhat longer (<60 ps) multiexponential process. This is indicative of multiple routes to the ground state.

Theoretical studies by Sobolowski and Domcke on 5,6-dihydroxyindole [5] suggests nonadiabatic hydrogen migration to form 6-Hydroxy-4-dihydro-indol-5-one (HHI) as a route to ultrafast internal conversion to the ground state. This is discussed in more detail in Chapter 2.

A study by Sundström and co-workers used femtosecond transient absorption of 5,6-dihydroxyindole in aqueous solution [15]. Following excitation at 266 nm they observed a 5-10 ps decay from the initially excited state to another excited state, then another 140-180 ps decay to the ground state, a triplet state and a cation state. These last two products may decay back to the ground state outside the timescale of the study.

Another later study by some of the same authors [20] used time resolved fluorescence after excitation of the solvated 5,6-dihydroxyindole at 266 and 280 nm. They observed two processes with timescales of 110 ps and 1.7 ns. The shorter process

dominated at shorter excitation wavelengths, and they suggested that this is equivalent to the 140 ps route they observed in their previous study (mentioned above, reference [15]). They suggest that in aqueous solution the cation with a solvated electron may be a route back to the ground state, and may be responsible for the longer process.

4- and 5-hydroxyindoles were studied by Ashfold and co-workers looking at total kinetic energy release of the photofragments, computational studies, and comparisons with 4- and 5-methoxyindole [9]. It was found that 4-hydroxyindole dissociated mainly along the O-H bond while 5-hydroxyindole dissociates mainly along the N-H bond.

Recent laser induced fluorescence of jet cooled 5-hydroxyindole by Schmitt and co-workers [14] reveal that different confirmers of 5-hydroxyindole have different ^1L_b lifetimes, possibly due to shifts in a conical intersection between the ππ* and the πσ*.

The studies summarised above show that there is a real interest in the photo-dynamics of these model biological systems, and that despite this there is still a lack of clarity about the different relaxation routes. A time-resolved photoelectron study of these important molecules could add greatly to the current understanding.

5.1.2. Models of the Adenine Sub-Unit of DNA

Adenine is one of the four bases that make up DNA and RNA, which is the basis for the genetic encoding and functioning of all known life on this planet. Ultraviolet photoexcitation of the DNA bases can lead to extremely dangerous structural changes [23] which can lead to damage of the genetic information, cell death, and cancer [24]. Understanding the effective self-protection mechanisms that DNA exhibits to protect itself from photo-excitation damage is important to understand how living organisms avoid harmful mutations. There has been much experimental and theoretical work on the UV photo dynamics of adenine (see refs. [25-27] and references within), both in the gas phase, as is discussed below, and in solution. Various studies have looked at adenine in the condensed phase including four-wave mixing [28] and 2D Fourier transform spectroscopy [29]. Absorption spectra in helium nano-droplets [30] reveal that the dynamics of both adenine and 9-methyladenine are affected by solvation. Significant line-broadening of the ^1L_b and $^1n\pi^*$ states suggest that the $^1n\pi^*/^1L_b$ conical intersection is more accessible in this phase.

There has been some controversy about the different possible routes of relaxation to the ground state in gas-phase adenine. The $^1\pi\sigma^*$ state in adenine is dissociative along

the N-H bond on the 5 membered ring [32] (see Figure 5.3a), and may play a significant role in the ultrafast internal conversion of the photo-excited bare adenine molecule via a further conical intersection with the ground state at extended N-H bond lengths [27, 33, 34]. These dynamics are similar to the molecules previously discussed in this thesis, and many other heterocyclic systems [35, 36]. This is the bond that, within the DNA strand, connects adenine to the sugar back bone (see Figure 5.3c). There are also another two radiationless routes to the ground state, via the $^1n\pi^*$ state [27, 33], and via the initially excited ^1L_a state [37]. These routes occur along ring puckering coordinates of the 6 membered ring [33, 37]. Time-resolved studies have observed decay on two time scales [34, 38-40]. One very fast process occurs in under 100 femtoseconds and a longer lived process decays on the order of a picosecond. Assigning these processes to relaxation routes has caused some controversy. It is important to understand not only how this molecule behaves in isolation, but also how it behaves when it is part of the larger biological unit. By replacing the NH bond that is responsible for the $^1\pi\sigma^*$ state's dissociative properties with a methyl group, this should change the energetic position of the $^1\pi\sigma^*$ state, and provide a closer model of the adenine nucleotide (Figure 5.3c).

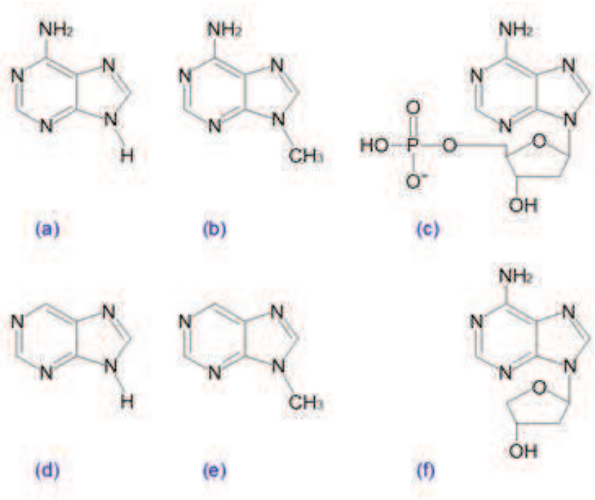


Figure 5.3: Model systems discussed in Section 5.1.2 (a) adenine, (b) 9-methyladenine, (c) adenine nucleotide - sub-unit of DNA, (d) purine, (e) 9-methylpurine and (f) 9-(2-tetrahydrofuanyl) purine. Note the 9-H tautomer shown in (a) is the dominant form in the gas phase [31].

Previous work in Stolow's group, in the same lab used for collecting the results shown in Chapter 2, used TRPES to probe the role of the $^1\pi\sigma^*$ state in gas-phase adenine and 9-methyladenine (Figure 5.3b) [40]. They observed that the photoelectron spectrum of the fast process looked qualitatively different, and the ratio of the fast process to the slow process was much decreased. They attribute the fast process to both the $^1\pi\sigma^*$ state and the ^1L_a state, and the long lived process to the $^1n\pi^*$ state. The interpretation of these results shows evidence that, at 267 nm excitation, electronic relaxation via the $^1\pi\sigma^*$ state is significant in adenine, but appears not to play a role once

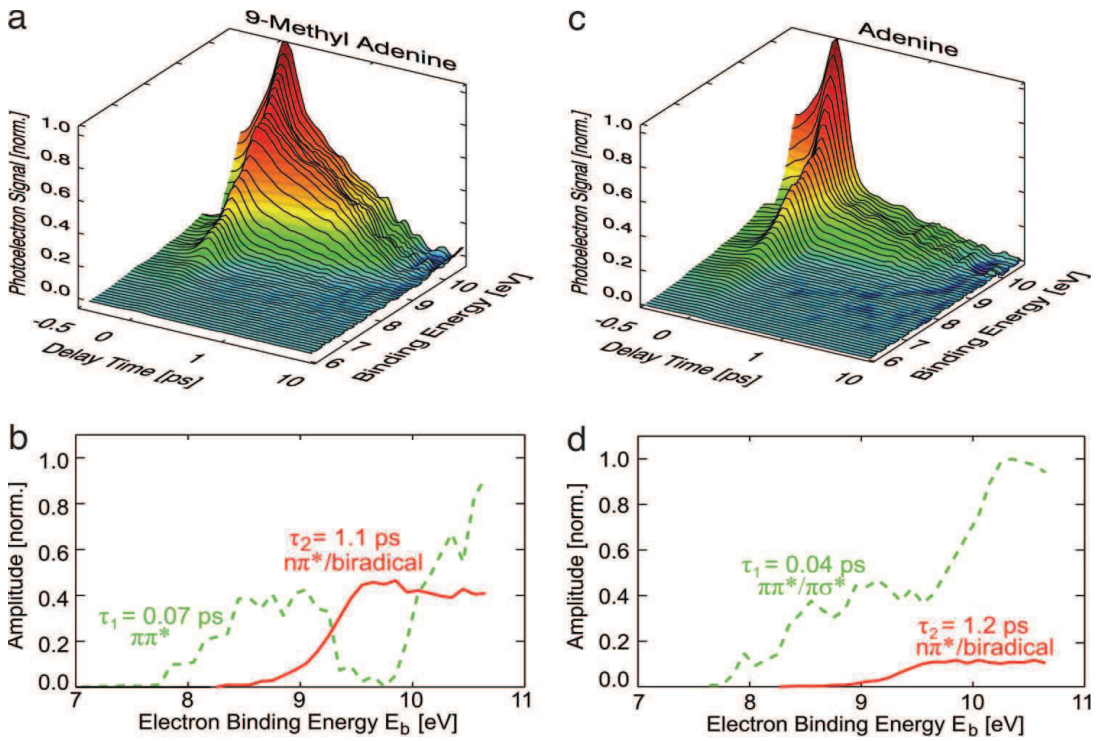


Figure 5.4 From [40]. Photoelectron spectra and deconvoluted decay-associated spectra (DAS). TRPES spectra for 9-methyl adenine (a) and adenine (c). The time dependence is plotted by using a linear-logarithmic scale with a linear scale in the region -0.5-1.0 ps and a logarithmic scale for delay times 1.0–10.0 ps. Below, (b) and (d) show the decomposition into decay-associated spectra obtained from the global fitting algorithm. The region between 9.3 and 10 eV of the τ_1 DAS (green) is associated with the $^1\pi\sigma^*$ state.

the 9N-H position is methylated. See Figure 5.4 and caption for more details. More recent time-resolved photoelectron spectroscopy on adenine at a range of wavelengths [39] agrees with this assignment and identifies an excitation energy cut-off of around 5.2 eV, below which the $^1\pi\sigma^*$ state is no longer populated.

Investigations into the influence of substituents other than simple methyl groups upon the electronic relaxation dynamics in these types of aromatic heterocyclic system have, up to now, been somewhat limited. Time-dependent density functional theory calculations by Alavi [41] have predicted, however, that the presence of an oxygen atom in some groups substituted at the 9N-H position in adenine introduces new low lying electronic states of $^1\pi\sigma^*$ character. The assumption that simple methylation of the N-H bond in adenine provides a suitable approximation of the sugar-phosphate backbone in DNA would therefore seem to be potentially unreliable.

Purine is the base of both adenine and guanine and as such research into purine will give insight into both these important biological molecules. A potential future use of the new time-resolved photoelectron imaging spectrometer described in Chapter 3 would be

to look at the photo-dynamics of purine and the purine derivatives shown in Figure 5.3d-f. The sugar group that forms the backbone of the DNA double helix strand may have a substantial effect on the photo-dynamics of the bases. In order to understand the effects that this may have, a stepwise approach may be taken by systematically adding various additional substituents, such as those shown in Figure 5.3d-f. These systems have already been synthesised by Magnus Bebbington in Heriot-Watt University, and, when studied using time resolved photoelectron spectroscopy, should help to reveal the effects that a substituent group with greater similarities to the actual biological unit would have on the photo-dynamics of this model biological system.

5.2. Molecular Sources

The molecular beam source currently employed is an excellent method of preparing the gas phase samples. It does, however, have some limitations when dealing with large, less volatile molecules. Detecting photoelectrons in a vacuum chamber allows the electrons to encounter no other molecules between the ionisation and detection events. This means that the kinetic energy the electron has when ejected from the molecule is retained and can be accurately detected. This results in highly differential measurements that can give energy-, time- and angle-resolved information to help reveal the dynamics of the molecule under study. The condensed phase is also important, however, as this is the phase in which the biological molecules are found in nature. This section discusses two alternatives for studying the photoelectron emissions of model biological systems in a vacuum chamber that could extend our setup and enable us to study new systems.

5.2.1. Liquid Microjet

The liquid microjet was first demonstrated by Hans and Kai Siegbahn in 1973, [42], using a formaldehyde jet to study dissolved potassium iodide. These experiments were limited to liquids with a low vapour pressure. The relatively high background pressures using these techniques requires a differentially pumped chamber, to allow the electrons to travel to the detector on a collision free path [43]. Faubel *et al.* were the first to demonstrate high vapour pressure supercooled liquid microjets in vacuum, studying the photoelectron spectroscopy of pure water [44, 45] using X-rays to perform the single photon ionisation, and a hemispherical analyser to record the photoelectron energies. High vapour pressure liquids can be studied if the surface area is reduced (jets between

5-20 μm diameter) [46] and the jets have a high flow velocity (between 30 and 120 m/s) [43]. The first technique helps avoid excess evaporation of the liquid, and the second creates a continually refreshed liquid source, ensuring that the sample at the target is never completely depleted. The liquid jet travels through the chamber to a liquid nitrogen cold-trap, where it freezes into filaments for later removal. As the jet travels it evaporatively cools, reaching a surface temperature of around 280 K for water, many times below the normal freezing temperature [47].

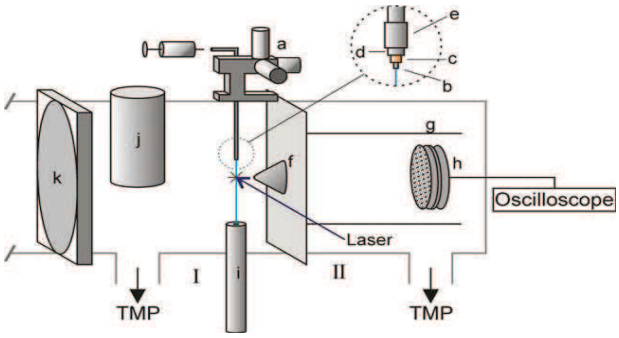


Figure 5.5: Schematic of the liquid microjet photoelectron spectrometer as implemented by Neumark's group. From [47].

This technique has mainly been used in X-ray photoelectron spectroscopy, however it has recently gained popularity in time-resolved photoelectron spectroscopy studies [48, 49]. At some point along the jet, usually about a couple of millimetres from the nozzle, it is intersected with the pump and probe laser beams. Figure 5.5 shows a schematic of a liquid jet photoelectron spectrometer as used by Neumark's group [47]. The background signal from the water in the jet can be subtracted by first running the experiment with pure water, and then with a solution of the molecule of interest. By looking at the difference between these spectra, the photoelectron spectra of just the molecule in solution can be extracted.

This method, although still in its infancy for time-resolved measurements, may prove to be very important for understanding the photoelectron spectroscopy of solvated molecules. It may also expand the possible range of molecules to be studied, as molecules that cannot enter the gas phase without breaking apart, such as some large biological molecules, could potentially be studied.

5.2.2. Laser Induced Acoustic Desorption

As discussed in Section 1.3.5 there are several methods for creating gas phase samples of low vapour pressure molecules. Matrix assisted laser desorption/ionisation (MALDI) [50, 51] ablates crystallised mixtures of the molecule of interest, matrix molecules and solvents, to produce a mixed gas phase plume. This is not suitable for

photoelectron studies as the sample is mixed with other species that contaminate the results. Electrospray Ionisation [86, 87] results in a mixture of highly charged molecules, often with protons either added or removed. This is of limited use for photoelectron studies as the molecules are not neutral, and are often a modified form of the molecule of interest.

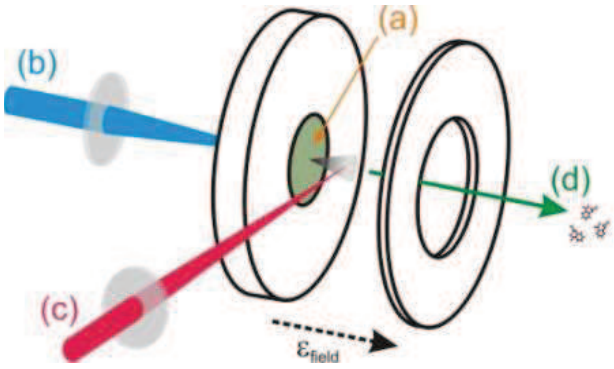


Figure 5.6: Overview of the LIAD approach, as discussed in the main text.

An alternative method for preparing molecular samples is Laser-Induced Acoustic Desorption (LIAD) [52-56]. In contrast to the techniques mentioned above, LIAD produces gas phase biomolecules that are free from any solvent effects or other contaminations. This technique can be applied to a wide range of low vapour pressure samples. It is also considerably cleaner and less destructive than simple laser ablation-based techniques where high energy laser pulses irradiate the sample directly. As depicted in Figure 5.6, a solid sample (a) is deposited on the surface of a thin metal foil. The other side of the foil is irradiated using a laser pulse (b). This interaction stimulates the propagation of an acoustic wave through the foil, resulting in sample molecules desorbing from the surface. Additional laser pulses (c) then interact with the desorbed neutral molecules, forming charged particles or fragmentation products (d) that may be subsequently extracted and analysed in an imaging experiment. The acoustic wave that is generated by the laser pulse has a frequency that is closer to the natural frequency of the molecule's surface bonds than to the intermolecular bonds. When the acoustic wave reaches the sample, it sets the molecules on the surface vibrating with respect to the surface, but not vibrating internally. This safely desorbs the molecule from the sample, with minimal risk of molecular fragmentation and can be used for molecules that are very fragile, or have very low vapour pressures.

Although the advantages of this scheme are clear for the ultrafast gas phase study of the dynamics of neutral biological systems [56], it suffers from the key limitations that the sample of interest is released into a large volume of low density. A possible solution to this problem would be to put the LIAD setup behind a molecular beam valve. This would allow the molecules to build up into a higher density gas, before being released

in a concentrated pulse through the valve, and become cooled through supersonic expansion into the chamber.

This technique would be ideal in assisting in the study of several important biological systems. Guanine has a very low vapour pressure [57] and so has few gas phase dynamics studies [58]. There are a few studies looking at the tautomers present in the gas phase using helium nanodroplets and laser ablation techniques [59, 60], and a large body of theoretical work (for example [61-63]). In addition, larger biological molecules like DNA and RNA nucleotides, larger sections of the eumelanin polymer, and some amino acids, can have very low vapour pressures and be thermally unstable. The LIAD technique in conjunction with our photoelectron spectrometer would be ideal to reveal more about the dynamics of these important systems.

5.3. Optical Sources

The choice of pump and probe wavelength is very important in revealing molecular dynamics. By tuning pump wavelengths to excite certain electronic states, a greater understanding of those particular states can be obtained. The choice of probe wavelength is also very important. Choosing a probe wavelength that does not initially excite the molecule is important, as otherwise there will be probe-pump dynamics (occurring in the negative time direction) as well as pump-probe dynamics. De-convoluting the different contributions to the final data set can be complicated. In addition, the probe must have enough energy to ionise not just the initially excited state, but preferably some of the lower lying states as well. This gives us a fuller dynamical picture of the molecular relaxation path. The duration of the pump and probe is another important consideration. Short pulses give good time resolution, but poor frequency resolution. Longer pulses can reveal more about the energies of the states involved. This section explores some of the possible options for extending the current experimental setup to allow a broader range of experiments to take place.

5.3.1. Frequency Domain Pulses

Our current setup allows the molecular dynamics to be explored with a very high time resolution, but does not give a very high energy resolution. If we were to implement a more monochromatic, longer pulse length laser in addition to our broad bandwidth, short pulse system, then we could conduct complementary experiments.

Studying the same molecule with both sources would mean that we could extract both high frequency resolution and high time resolution results. One source of such monochromatic light pulses is a dye laser [64, 65]. These lasers produce pulses in the picosecond or nanosecond time range with very narrow bandwidth. An advantage of these systems is that they are tuneable over a broad range of wavelengths, including the UV range that we are interested in for the model biological systems that we want to study. The laser wavelength can be scanned automatically to produce resonantly enhanced multi-photon ionisation (REMPI) spectra of the jet cooled molecules [66]. With this setup we could obtain gas phase spectra of the jet cooled molecules we want to study, and extract ionisation potentials. This complementary information is important in planning our experiments and interpreting our experimental results. Often the gas phase absorption spactra or the ionisation potentials of the systems we wish to study are not well known, or do not span the region of interest. This can prove problematic in the analysis of data and the choice of initial pump and probe wavelengths. With this setup we could determine easily these values ourselves. It would not require any changes to the spectrometer setup but would require a few changes to the optical setup to allow the new laser to be implemented into the current system. The complex optical setup described in Section 3.1 for the Ti:Sapphire femtosecond system is primarily to change the wavelength from infrared to ultraviolet. The changes required to implement a dye laser system would be minimal because the laser is tuneable over a wide range of frequencies and often contains a second harmonic setup built into the laser, therefore the only optics required would be to guide the beam into the spectrometer.

5.3.2. Tuneable Ultraviolet Pulses

In order to enhance the flexibility of the ultrafast optical system, thus increasing the range of experiments we could conduct, another optical parametric amplifier would be very beneficial. At the moment we are using one optical parametric amplifier and the other UV pulses are created by doubling and mixing the laser output as described in Section 3.1.4. This latter process is not very flexible, as the possible wavelengths are restricted to harmonics of the laser fundamental. These processes can give us 400 nm, 267 nm, and 200 nm pulses. By tuning the output wavelength of the laser the wavelengths can be shifted slightly, but this is a time consuming and difficult process. In order to run more expansive studies on model biological systems, we would ideally

like two sources that can both be easily tuned over a wide wavelength range. There are several reasons for this. The pump beam should be tuneable as we often wish to excite above and below specific barriers, so we require specific photon energies. In addition, to fully understand the dynamics of a system it is advisable to run the same experiment with several different pump wavelengths. This can reveal much more about the dynamics of a molecule, including thresholds for various processes, and the different pathways for different excited states. In addition to this, the probe wavelength should also be flexible. If the probe pulse is absorbed by the ground state molecule, then the pump pulse may be able to ionise the resultant excited state. This is not the process we want to look at, and it can often mask fast dynamics occurring from the desired pump-probe signals. To avoid this problem, a probe wavelength should be chosen that is both not strongly absorbed by the molecule and energetic enough to ionise molecules in the excited states of interest (see the next section for a discussion on the advantages of high energy probe photons). In order to choose this wavelength a jet-cooled gas-phase absorption spectrum is extremely useful. Some molecules do not have published spectra, but the previous section outlines a way of obtaining them. If there is only one tuneable source, then the scope and range of possible experiments are restricted.

Since the very first demonstrations of an optical parametric amplifier, only five years after the invention of the laser [67], these systems have developed into reliable sources that, with subsequent non-linear processes, can be tuned from 20 μm all the way to 190 nm [67-71] (and possibly even shorter, as discussed in the next section). The state-of-the-art TOPAS optical parametric amplifier [72] is a computer controlled optical parametric amplifier that allows the user to quickly and easily tune the ultrafast pulses to the desired photon energy. Implementing this system into our current setup would minimise the turn-around time between experiments and allow a wider range of experiments to be conducted.

5.3.3. Vacuum Ultraviolet Generation

Although the methods described in this thesis are extremely powerful in yielding deep insight into the complex photo-physics of many different molecular systems [73, 74]. There are often limitations due to the fact that the experimenter cannot see the complete ‘path’ along the reaction coordinate all the way from reactants to products. Since no rigid selection rules govern ionisation processes, all electronic states

participating in the system dynamics may, in principle, be directly observed. Different states are projected onto different regions of the ionisation continuum and, due to energy conservation, therefore show their dynamical involvement in different regions of the photoelectron spectrum leading to highly differential measurement of the underlying photophysics. However, due to restrictions imposed by a combination of the Franck-Condon principle, explained in Chapter 1, and the available energy in the probe photon, it is not always possible to observe all the populated electronic states, especially those that are lower lying.

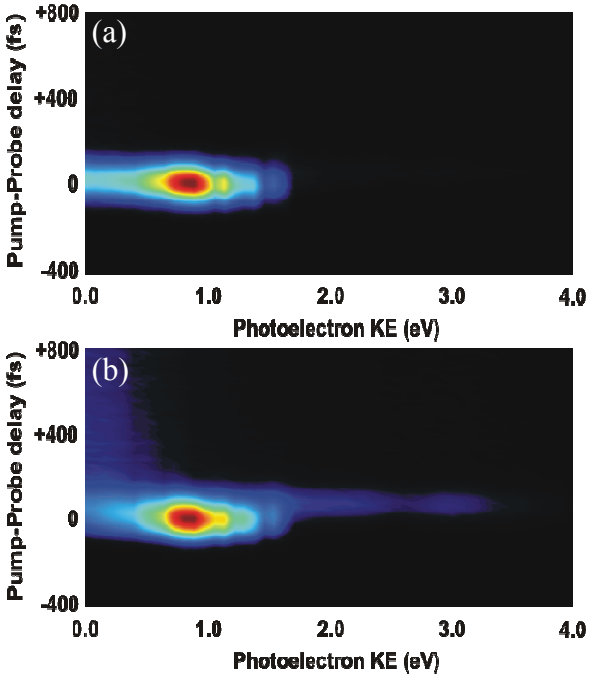


Figure 5.7: 1-3 butadiene time-resolved photoelectron spectroscopy data, displayed in a colour-map intensity plot. (a) displays the data taken at low probe intensity, and (b) with high probe intensity.

In order to illustrate this, a previous study by Townsend and Stolow considered 1,3-butadiene TRPES data, following excitation to the S_2 electronic state with a 20 nJ, 200 nm pump pulse and subsequent ionisation with a 350 nJ, 267 nm probe. Figure 5.7a shows that at low probe intensities the time-resolved photoelectron spectrum only exhibits a rapid (< 100 fs) decay. This is due to the decay of the initially prepared S_2 state via non-adiabatic coupling. Although the 267 nm probe photon has sufficient energy to ionise the state into which the S_2 population is transferred (the S_1 excited electronic state) the probability for this ionisation to occur is negligibly small owing to the poor Franck-Condon overlap between the highly vibrationally excited S_1 and the energetically accessible states of the cation ground state D_0 (as illustrated in Figure 5.8, labelled (a)). Because of this, no significant photoelectron signal is observed from the S_1 state, leading to a reduced 'view' along the reaction coordinates.

Increasing the 267 nm probe energy until there is significant two-photon ionisation occurring from the excited states can overcome this problem. This allows ionisation to higher lying ionic states, where the Franck-Condon overlap is greatly improved (demonstrated in Figure 5.8 labelled (b)). Figure 5.7b shows the experimental results

using this method. A much more complete 'view' along the reaction coordinate is now produced, with the ultrafast decay of the S_1 state now visible (in the 1.7 – 3.4 eV region of the spectrum). In addition to this, the appearance of long-lived signal that extends beyond 800 fs is observed (between 0 – 0.5 eV). This is produced from ionisation from the vibrationally excited S_0 ground state, which is populated following the decay of S_1. The view along the reaction coordinate now fully spans the evolution from reactants to product, and the complete dynamical path is revealed.

The spectral features observed due to this two-photon process seen in Figure 5.7b are much weaker than the one-photon features because of the reduced cross-section for two-photon ionisation. The excellent signal to noise given by the large absorption cross section of 1,3-butadiene permits this process to be observed in this simple example. The more general use of this scheme is often hindered by the very low signal levels that are produced from a two-photon scheme. This can be particularly challenging when photoelectron signal from the two-photon and one-photon processes are not spectrally resolved from each other.

These problems can all be avoided by using a very high energy photon, as demonstrated in Figure 5.8 labelled (c), instead of two lower energy photons. Femtosecond vacuum ultraviolet (VUV) pulses can offer these high energy photons. VUV (40 - 190 nm) is defined as the UV

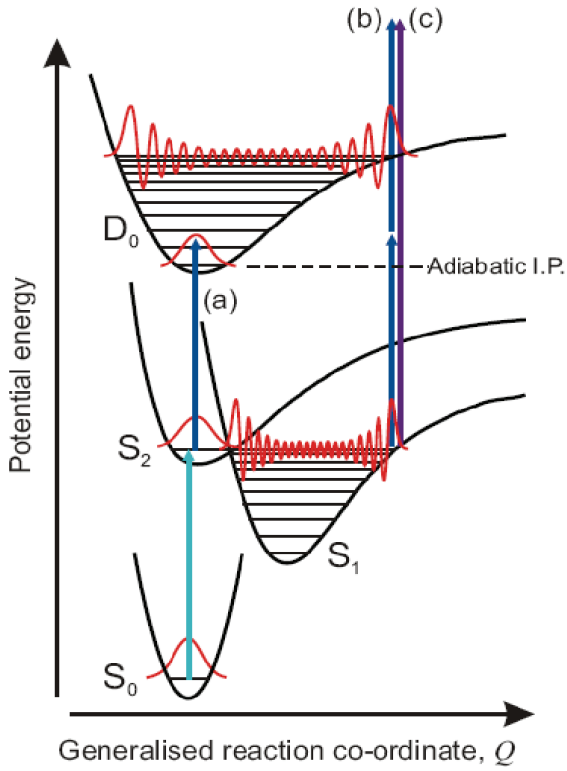


Figure 5.8: Cartoon depiction of the role of the Franck-Condon principle in TRPES. Schematic vibrational wavefunctions are shown in red. For simplicity, a stationary state representation is used. In scheme (a) there is strong ionisation from S_2 but negligible ionisation from S_1 owing to the very poor vibrational wavefunction overlap with the low-lying cation states. In scheme (b) the use of two-photon ionisation opens up higher lying states in the cation for which the vibrational wavefunction overlap with S_1 is greatly improved. The signal from S_1 may be further enhanced, to a much larger extent, using a single VUV photon – scheme (c).

wavelength range that is absorbed by air. It is difficult to generate as it cannot propagate in a standard lab environment. These pulses would enable the non-adiabatic dynamics in a wide range of photochemical systems to be observed fully via a one-photon scheme. This probe should ideally be broadly tuneable across the VUV region, so that the energy may be tailored for a given biological system. This would allow us to maximise the view along the reaction co-ordinate without ground state ionisation obscuring the results, and would potentially provide a complete 'map' of all dynamical information of the photochemical processes. This information would yield previously unobtainable levels of insight into the pathways that allow excess energy redistribution in many chemically and biologically important systems.

Recently, four-wave difference-frequency mixing in a hollow-core argon-filled waveguide has been used to generate VUV femtosecond pulses [75, 76]. This approach used the 800 nm fundamental frequency from a Ti:Sapphire system, along with the 267 nm third harmonic, to produce 160 nm light. When the input pulse energy is a few hundred microjoules, a few hundred nanojoules of VUV is generated, with sub-100 fs duration. These intensities are well-suited for a single photon probe step, as outlined above, and the input energies are easily achievable with our current setup. By varying the Ti:Sapphire output, limited tuning (168-156 nm) of the VUV is possible, however ideally we would want to extend this range to shorter wavelengths. This could be done by replacing the fundamental beam with the tuneable signal beam output from an OPA source that is detailed in Sections 3.1.5 and 5.3.2. Difference- and sum-frequency mixing this with the third harmonic beam would enable tuneable VUV femtosecond pulses to be generated down to 110 nm. The VUV output from this process would fully span the 110 nm – 165 nm photon wavelength region, which is ideally matched for tuning to just below the ionisation potential of many of the small model biological systems of interest (for example those discussed in Section 5.1). A VUV system attached to our current setup would allow us to reveal deeper insights by seeing further along the photoexcitation reaction path of these important biological systems.

5.4. Conclusions

This thesis aims to provide an overview of the work that has been undertaken over the last four years. A large proportion of time and effort went into designing, implementing, and troubleshooting the various aspects of the equipment and software

that was installed into the lab at Heriot-Watt University. These have been described in Chapter 3. I was also privileged to undertake an ultrafast molecular dynamics investigation on models of the eumelanin pigment. One of these was conducted in Canada at the NRC (Chapter 2), with the support of Albert Stolow and Oliver Schalk. Building on this, the other was on the newly commissioned equipment previously mentioned, in Heriot-Watt University (Chapter 4).

In looking to understand the photo-dynamics of 5,6-dihydroxyindole, which is a constituent part of the eumelanin polymer, we initially looked at indole and 5-hydroxyindole to investigate the effect of the additional hydroxyl group. We found that the molecules exhibited very similar dynamics, and concluded that in 5-hydroxyindole the hydroxyl group is not involved in the photo-dynamics at the wavelengths we used. We then went on to study phenol and its derivatives, as catechol is a good model for 5,6-dihydroxyindole in some regards. Our aim was to understand how two hydroxyl groups will affect the dynamics of an aromatic ring, and the effect of varying the relative proximity of the two groups. It was found that when the two groups were in close proximity to each other (catechol), the dynamics were very different from when the groups were isolated from each other (resorcinol and hydroquinone), or there was only one group present (phenol). In catechol's case we saw a dramatic increase in relaxation rate. This suggests that 5,6-dihydroxyindole, which has two hydroxyl groups in close proximity, may have dissimilar dynamics to both 5-hydroxyindole and 6-hydroxyindole. Finally, we explored several methods of taking these studies forward, and improvements to the experimental setup.

During the course of this PhD I have gained many skills and much knowledge about the experimental techniques and the analysis involved in the time resolved photoelectron spectroscopy of model biological systems. I have gained experience with the installation, alignment, and optimisation of both femtosecond pulsed lasers and the related nonlinear optics required to change the wavelength of these pulses. The design and troubleshooting of two vacuum systems, high pressure gas systems, and the various components that make successful photoelectron spectrometers has given me many new skills. The development of the software required to run the experiment and process and analyse the resultant data has vastly improved my programming capabilities. Finally, the basic understanding of chemical techniques and processes, including molecular dynamics and spectroscopy, will be invaluable for my future.

5.5. References

[1] P. Meredith and T. Sarna, *The Physical and Chemical Properties of Eumelanin*, Pigm. Cell. Res., **19**, 572, (2006).

[2] N. Sundaraganesan, H. Umamaheswari, B. D. Joshua, C. Meganathan, and M. Ramalingam, *Molecular Structure and Vibrational Spectra of Indole and 5-Aminoindole by Density Functional Theory and Ab Initio Hartree-Fock Calculations*, J. Mol. Struct. THEOCHEM, **850**, 84, (2008).

[3] D. Robinson, N. A. Besley, E. A. M. Lunt, P. O'Shea, and J. D. Hirst, *Electronic Structure of 5-Hydroxyindole: From Gas Phase to Explicit Solvation*, J. Phys. Chem. B, **113**, 2535, (2009).

[4] M. Kubota and T. Kobayashi, *Electronic Structures of Melatonin and Related Compounds Studied by Photoelectron Spectroscopy*, J. Elec. Spec., **128**, 165, (2003).

[5] A. L. Sobolewski and W. Domcke, *Photophysics of Eumelanin: Ab Initio Studies on the Electronic Spectroscopy and Photochemistry of 5,6-Dihydroxyindole*, ChemPhysChem, **8**, 756, (2007).

[6] Y. V. Il'ichev and J. D. Simon, *Building Blocks of Eumelanin: Relative Stability and Excitation Energies of Tautomers of 5,6-Dihydroxyindole and 5,6-Indolequinone*, J. Phys. Chem. B, **107**, 7162, (2003).

[7] R. Livingstone, O. Schalk, A. E. Boguslavskiy, G. R. Wu, L. T. Bergendahl, A. Stolow, M. J. Paterson, and D. Townsend, *Following the Excited State Relaxation Dynamics of Indole and 5-Hydroxyindole Using Time-Resolved Photoelectron Spectroscopy*, J. Chem. Phys., **135**, 194307, (2011).

[8] K. J. Lawrie, P. Meredith, and R. P. McGeary, *Synthesis and Polymerization Studies of Organic-Soluble Eumelanins*, Photochem. Photobiol., **84**, 632, (2008).

[9] T. A. A. Oliver, G. A. King, and M. N. R. Ashfold, *Position Matters: Competing O-H and N-H Photodissociation Pathways in Hydroxy- and Methoxy-Substituted Indoles*, Phys. Chem. Chem. Phys., **13**, 14646, (2011).

[10] M. L. Tran, B. J. Powell, and P. Meredith, *Chemical and Structural Disorder in Eumelanins: A Possible Explanation for Broadband Absorbance*, Biophys. J., **90**, 743, (2006).

[11] P. Meredith, B. J. Powell, J. Riesz, S. P. Nighswander-Rempel, M. R. Pederson, and E. G. Moore, *Towards Structure-Property-Function Relationships for Eumelanin*, Soft Matter, **2**, 37, (2006).

[12] Y. H. Huang and M. Sulkes, *Anomalously Short Fluorescence Lifetimes in Jet Cooled 4-Hydroxyindole. Evidence for Excited State Tautomerism and Proton Transfer in Clusters*, Chem. Phys. Lett., **254**, 242, (1996).

[13] M. Sulkes, Y. H. Huang, and L. Byers, *Anomalously Short Fluorescence Lifetimes in Jet Cooled 4-Hydroxyindole*, Abstr. Pap. Am. Chem. S., **211**, 54, (1996).

[14] O. Oeltermann, C. Brand, M. Wilke, and M. Schmitt, *Ground and Electronically Excited Singlet State Structures of the syn and anti Rotamers of 5-Hydroxyindole*, J. Phys. Chem. A, **116**, 7873, (2012).

[15] M. Gauden, A. Pezzella, L. Panzella, A. Napolitano, M. d'Ischia, and V. Sundström, *Ultrafast Excited State Dynamics of 5,6-Dihydroxyindole, A Key Eumelanin Building Block: Nonradiative Decay Mechanism*, J. Phys. Chem. B, **113**, 12575, (2009).

[16] J. B. Nofsinger, T. Ye, and J. D. Simon, *Ultrafast Nonradiative Relaxation Dynamics of Eumelanin*, J. Phys. Chem. B, **105**, 2864, (2001).

[17] J. E. de Albuquerque, C. Giacomantonio, A. G. White, and P. Meredith, *Study of Optical Properties of Electropolymerized Melanin Films by Photopyroelectric Spectroscopy*, Eur. Biophys. J. Biophy., **35**, 190, (2006).

[18] J. Riesz, J. Gilmore, and P. Meredith, *Quantitative Scattering of Melanin Solutions*, Biophys. J., **90**, 4137, (2006).

[19] J. Riesz, T. Sarna, and P. Meredith, *Radiative Relaxation in Synthetic Pheomelanin*, J. Phys. Chem. B, **110**, 13985, (2006).

[20] A. Huijser, A. Pezzella, and V. Sundstrom, *Functionality of Epidermal Melanin Pigments: Current Knowledge on UV-Dissipative Mechanisms and Research Perspectives*, Phys. Chem. Chem. Phys., **13**, 9119, (2011).

[21] J. Catalán, P. Perez, and A. U. Acuña, *Indole Spectroscopy - the Location of the ^1L_a and ^1L_b Electronic States and the Absorption-Spectrum*, J. Mol. Struct., **142**, 179, (1986).

[22] X. Meng, T. Harricharran, and L. J. Juszczak, *A Spectroscopic Survey of Substituted Indoles Reveals Consequences of a Stabilized ^1L_b Transition*, Photochem. Photobiol., Accepted, (2012).

[23] I.-R. Lee, W. Lee, and A. H. Zewail, *Primary Steps of the Photoactive Yellow Protein: Isolated Chromophore Dynamics and Protein Directed Function*, P. Natl. Acad. Sci. USA, **103**, 258, (2006).

[24] G. P. Pfeifer, Y.-H. You, and A. Besaratinia, *Mutations Induced by Ultraviolet Light*, Mutation Res., **571**, 19, (2005).

[25] M. Barbatti and S. Ullrich, *Ionization Potentials of Adenine Along the Internal Conversion Pathways*, Phys. Chem. Chem. Phys., **13**, 15492, (2011).

[26] M. Barbatti, Z. Lan, R. Crespo-Otero, J. J. Szymczak, H. Lischka, and W. Thiel, *Critical Appraisal of Excited State Nonadiabatic Dynamics Simulations of 9H-Adenine*, J. Chem. Phys., **137**, 22A503, (2012).

[27] C. Z. Bisgaard, H. Satzger, S. Ullrich, and A. Stolow, *Excited-State Dynamics of Isolated DNA Bases: A Case Study of Adenine*, ChemPhysChem, **10**, 101, (2009).

[28] B. A. West, J. M. Womick, and A. M. Moran, *Probing Ultrafast Dynamics in Adenine With Mid-UV Four-Wave Mixing Spectroscopies*, J. Phys. Chem. A, **115**, 8630, (2011).

[29] C.-h. Tseng, P. Sándor, M. Kotur, T. C. Weinacht, and S. Matsika, *Two-Dimensional Fourier Transform Spectroscopy of Adenine and Uracil Using Shaped Ultrafast Laser Pulses in the Deep UV*, J. Phys. Chem. A, **116**, 2654, (2011).

[30] S. Smolarek, A. M. Rijs, W. J. Buma, and M. Drabbels, *Absorption Spectroscopy of Adenine, 9-Methyladenine, and 2-Aminopurine in Helium Nanodroplets*, Phys. Chem. Chem. Phys., **12**, 15600, (2010).

[31] G. C. P. van Zundert, S. Jaeqx, G. Berden, J. M. Bakker, K. Kleinermanns, J. Oomens, and A. M. Rijs, *IR Spectroscopy of Isolated Neutral and Protonated Adenine and 9-Methyladenine*, ChemPhysChem, **12**, 1921, (2011).

[32] M. G. D. Nix, A. L. Devine, B. Cronin, and M. N. R. Ashfold, *Ultraviolet Photolysis of Adenine: Dissociation via the* $^1\pi\sigma^*$ *State*, J. Chem. Phys., **126**, 124312, (2007).

[33] W. M. I. Hassan, W. C. Chung, N. Shimakura, S. Koseki, H. Kono, and Y. Fujimura, *Ultrafast Radiationless Transition Pathways through Conical Intersections in Photo-Excited 9H-Adenine*, Phys. Chem. Chem. Phys., **12**, 5317, (2010).

[34] K. L. Wells, G. M. Roberts, and V. G. Stavros, *Dynamics of H-Loss in Adenine Via the $^1\pi\sigma^*$ State Using a Combination of ns and fs Laser Spectroscopy*, Chem. Phys. Lett., **446**, 20, (2007).

[35] A. L. Sobolewski, W. Domcke, C. Dedonder-Lardeux, and C. Jouvet, *Excited-State Hydrogen Detachment and Hydrogen Transfer Driven by Repulsive $^1\pi\sigma^*$ States: A New Paradigm for Nonradiative Decay in Aromatic Biomolecules*, Phys. Chem. Chem. Phys., **4**, 1093, (2002).

[36] M. N. R. Ashfold, G. A. King, D. Murdock, M. G. D. Nix, T. A. A. Oliver, and A. G. Sage, *$\pi\sigma^*$ Excited States in Molecular Photochemistry*, Phys. Chem. Chem. Phys., **12**, 1218, (2010).

[37] I. Conti, M. Garavelli, and G. Orlandi, *Deciphering Low Energy Deactivation Channels in Adenine*, J. Am. Chem. Soc., **131**, 16108, (2009).

[38] C. Canuel, M. Mons, F. Piuzzi, B. Tardivel, I. Dimicoli, and M. Elhanine, *Excited States Dynamics of DNA and RNA Bases: Characterization of a Stepwise Deactivation Pathway in the Gas Phase*, J. Chem. Phys., **122**, 074316, (2005).

[39] N. L. Evans and S. Ullrich, *Wavelength Dependence of Electronic Relaxation in Isolated Adenine Using UV Femtosecond Time-Resolved Photoelectron Spectroscopy*, J. Phys. Chem. A, **114**, 11225, (2010).

[40] H. Satzger, D. Townsend, M. Z. Zgierski, S. Patchkovskii, S. Ullrich, and A. Stolow, *Primary Processes Underlying the Photostability of Isolated DNA Bases: Adenine*, P. Natl. Acad. Sci. USA, **103**, 10196, (2006).

[41] S. Alavi, *Simple Ethers as Models of Sugar Molecules in Calculations of Vertical Excitation Energies of DNA and RNA Nucleosides*, J. Phys. Chem. A, **109**, 9536, (2005).

[42] H. Siegbahn and K. Siegbahn, *ESCA Applied to Liquids*, J. Elec. Spec., **2**, 319, (1973).

[43] B. Winter and M. Faubel, *Photoemission from Liquid Aqueous Solutions*, Chem. Rev., **106**, 1176, (2006).

[44] M. Faubel, B. Steiner, and J. P. Toennies, *Photoelectron Spectroscopy of Liquid Water, Some Alcohols, and Pure Nonane in Free Micro Jets*, J. Chem. Phys., **106**, 9013, (1997).

[45] M. Faubel, S. Schlemmer, and J. P. Toennies, *A Molecular Beam Study of the Evaporation of Water from a Liquid Jet*, Z. Phys. D: At. Mol. Clusters, **10**, 269, (1988).

[46] B. Winter, *Liquid Microjet for Photoelectron Spectroscopy*, Nucl. Instrum. Meth. A, **601**, 139, (2009).

[47] A. T. Shreve, T. A. Yen, and D. M. Neumark, *Photoelectron Spectroscopy of Hydrated Electrons*, Chem. Phys. Lett., **493**, 216, (2010).

[48] Y. Tang, Y.-i. Suzuki, H. Shen, K. Sekiguchi, N. Kurahashi, K. Nishizawa, P. Zuo, and T. Suzuki, *Time-Resolved Photoelectron Spectroscopy of Bulk Liquids at Ultra-Low Kinetic Energy*, Chem. Phys. Lett., **494**, 111, (2010).

[49] O. Link, E. Lugovoy, K. Siefermann, Y. Liu, M. Faubel, and B. Abel, *Ultrafast Electronic Spectroscopy for Chemical Analysis near Liquid Water Interfaces: Concepts and Applications*, Appl. Phys. A, **96**, 117, (2009).

[50] R. E. Russo, X. Mao, H. Liu, J. Gonzalez, and S. S. Mao, *Laser Ablation in Analytical Chemistry - a Review*, Talanta, **57**, 425, (2002).

[51] M. N. R. Ashfold, F. Claeyssens, G. M. Fuge, and S. J. Henley, *Pulsed Laser Ablation and Deposition of Thin Films*, Chem. Soc. Rev., **33**, 23, (2004).

[52] V. V. Golovlev, S. L. Allman, W. R. Garrett, N. I. Taranenko, and C. H. Chen, *Laser-Induced Acoustic Desorption*, Int. J. Mass. Spec., **169**, 69, (1997).

[53] J. Pérez, L. E. Ramírez-Arizmendi, C. J. Petzold, L. P. Guler, E. D. Nelson, and H. I. Kenttämaa, *Laser-Induced Acoustic Desorption/Chemical Ionization in Fourier-Transform Ion Cyclotron Resonance Mass Spectrometry*, Int. J. Mass. Spec., **198**, 173, (2000).

[54] R. C. Shea, C. J. Petzold, J. Liu, and H. I. Kenttämaa, *Experimental Investigations of the Internal Energy of Molecules Evaporated via Laser-Induced Acoustic Desorption into a Fourier Transform Ion Cyclotron Resonance Mass Spectrometer*, Anal. Chem., **79**, 1825, (2007).

[55] I. Bald, I. Dąbkowska, and E. Illenberger, *Probing Biomolecules by Laser–Induced Acoustic Desorption: Electrons at Near Zero Electron Volts Trigger Sugar–Phosphate Cleavage*, Angew. Chem. Int. Edit., **47**, 8518, (2008).

[56] C. R. Calvert, L. Belshaw, M. J. Duffy, O. Kelly, R. B. King, A. G. Smyth, T. J. Kelly, J. T. Costello, D. J. Timson, W. A. Bryan, T. Kierspel, P. Rice, I. C. E. Turcu, C. M. Cacho, E. Springate, I. D. Williams, and J. B. Greenwood, *LIAD-fs*

Scheme for Studies of Ultrafast Laser Interactions with Gas Phase Biomolecules, Phys. Chem. Chem. Phys., **14**, 6289, (2012).

[57] A. L. F. de Barros, A. Medina, F. Zappa, J. M. Pereira, E. Bessa, M. H. P. Martins, L. F. S. Coelho, W. Wolff, and N. V. de Castro Faria, *A Simple Experimental Arrangement for Measuring the Vapour Pressures and Sublimation Enthalpies by the Knudsen Effusion Method: Application to DNA and RNA Bases*, Nucl. Instrum. Meth. A, **560**, 219, (2006).

[58] S. Ullrich, T. Schultz, M. Z. Zgierski, and A. Stolow, *Electronic Relaxation Dynamics in DNA and RNA Bases Studied by Time-Resolved Photoelectron Spectroscopy*, Phys. Chem. Chem. Phys., **6**, 2796, (2004).

[59] M. Mons, F. Piuzzi, I. Dimicoli, L. Gorb, and J. Leszczynski, *Near-UV Resonant Two-Photon Ionization Spectroscopy of Gas Phase Guanine: Evidence for the Observation of Three Rare Tautomers*, J. Phys. Chem. A, **110**, 10921, (2006).

[60] I. Pena, V. Vaquero, J. Lopez, and J. Alonso, *Probing Guanine and Cytosine Tautomers in the Gas Phase*, International Symposium On Molecular Spectroscopy, **64**, (2009).

[61] H. Chen and S. Li, *Ab Initio Study on Deactivation Pathways of Excited 9H-Guanine*, J. Chem. Phys., **124**, 154315, (2006).

[62] Z. Lan, E. Fabiano, and W. Thiel, *Photoinduced Nonadiabatic Dynamics of 9H-Guanine*, ChemPhysChem, **10**, 1225, (2009).

[63] J. Cerny, V. Spirko, M. Mons, P. Hobza, and D. Nachtigallova, *Theoretical Study of the Ground and Excited States of 7-Methyl Guanine and 9-Methyl Guanine: Comparison with Experiment*, Phys. Chem. Chem. Phys., **8**, 3059, (2006).

[64] B. Soffer and B. McFarland, *Continuously Tunable, Narrow-Band Organic Dye Lasers*, Appl. Phys. Lett., **10**, 266, (1967).

[65] T. W. Hänsch, *Repetitively Pulsed Tunable Dye Laser for High Resolution Spectroscopy*, Appl. Opt., **11**, 895, (1972).

[66] E. Nir, M. Müller, L. I. Grace, and M. S. de Vries, *REMPI Spectroscopy of Cytosine*, Chem. Phys. Lett., **355**, 59, (2002).

[67] S. Akhmanov, A. Kovrigin, A. Piskarskas, V. Fadeev, and R. Khokhlov, *Observation of Parametric Amplification in the Optical Range*, J. Exp. Theor. Phys. Lett., **2**, 191, (1965).

[68] M. K. Reed, M. K. Steiner-Shepard, and D. K. Negus, *Widely Tunable Femtosecond Optical Parametric Amplifier at 250 kHz with a Ti:Sapphire Regenerative Amplifier*, Opt. Lett., **19**, 1855, (1994).

[69] M. K. Reed, M. K. Steiner-Shepard, M. S. Armas, and D. K. Negus, *Microjoule-Energy Ultrafast Optical Parametric Amplifiers*, J. Opt. Soc. Am. B, **12**, 2229, (1995).

[70] K. R. Wilson and V. V. Yakovlev, *Ultrafast Rainbow: Tunable Ultrashort Pulses from a Solid-State Kilohertz System*, J. Opt. Soc. Am. B, **14**, 444, (1997).

[71] G. Cerullo, M. Nisoli, S. Stagira, and S. De Silvestri, *Sub-8-fs Pulses from an Ultrabroadband Optical Parametric Amplifier in the Visible*, Opt. Lett., **23**, 1283, (1998).

[72] Coherent_Inc, *TOPAS-800-fs Optical Parametric Amplifier*, MC-039-06-0M0512 Rev.C, (2012).

[73] A. Stolow, A. E. Bragg, and D. M. Neumark, *Femtosecond Time-Resolved Photoelectron Spectroscopy*, Chem. Rev., **104**, 1719, (2004).

[74] A. Stolow and J. G. Underwood, *Time-Resolved Photoelectron Spectroscopy of Nonadiabatic Dynamics in Polyatomic Molecules*, Adv. Chem. Phys., 497, (2008).

[75] P. Tzankov, O. Steinkellner, J. Zheng, M. Mero, W. Freyer, A. Husakou, I. Babushkin, J. Herrmann, and F. Noack, *High-Power Fifth-Harmonic Generation of Femtosecond Pulses in the Vacuum Ultraviolet Using a Ti:Sapphire Laser*, Opt. Express., **15**, 6389, (2007).

[76] M. Beutler, M. Ghotbi, F. Noack, and I. V. Hertel, *Generation of Sub-50-fs Vacuum Ultraviolet Pulses by Four-Wave Mixing in Argon*, Opt. Lett., **35**, 1491, (2010).

Appendices – Computer Code

Appendix.A. Analyse Data

```
function varargout = Analyse_Data_1_0(varargin)
%Analyses and fits Data processed by Process_Data_1_0
% Help:
%To use plotE run the program. Load the required processed.mat file to evaluate and
%play :o) .dat files can also be loaded, but only to view and/or re-export.
%When you move to KE view you will need to load a calibration file. It
%will suggest a default file for you to use.\n
%Once you are finished with the file then export it into a format
%recognisable by Helmut's program using "Export File" button. You can view
%the exported files using "LOAD FILE" button. If you wish to add two
%datasets, first export them both with the same energy min, max and
%spacing. Make sure you are happy with the settings you've chosen, as most
%of them you can't change later.Then press the
%"ADD FILE" button and choose the two files you have exported. If the
%x-offset has changed use the controls to find the new offset, then click
%"Add Files" to create the new added file. You can view
%different summations of sections of the data using the sliders at the
%sides of the graph. If you want to save a section, click the relevent
%"Snap" button (bottom left hand side). Once you are finished hit the
%small "Export" button and a .txt file will open with all the saved
%sections and the current section.\n
%You can export the figures as a tif file using "Save figure". This will
%open a new window with the figure you ask for. Once you are happy with
%how this figure looks, click Save and your tif file will be saved.\n
%The imagegraph displayed at the top right is the summed image of all delays
%inbetween the two delays slider positions. This image is Abel
%Transformed, not the raw data image. You can also see the image in polar
%coordinates by clicking the tick box immediately below it. When the Z
%Axis is in "Display Fit" mode, only one delay is looked at at any one
%time. In this mode the image displayed is the image at that delay time,
%not a summed image.\n
%The colormap is displayed to the left of the window. Please note, this is
%the colormap for the 3D main intensity graph. Also, please don't confuse
%the numbers on the colormap as the numbers for the Delay Time Side Graph axis.
%Have fun :o)

% Last Modified by GUIDE v2.5 15-May-2012 19:47:27

%% Begin initialization code - DO NOT EDIT
gui_Singleton = 0;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Analyse_Data_1_0_OpeningFcn, ...
                   'gui_OutputFcn',  @Analyse_Data_1_0_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
%% Executes just before Analyse_Data_1_0 is made visible.
function Analyse_Data_1_0_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;%set up handles.etc

%set the initial values
load start; start=0; save('start','start')              %#ok<NASGU>
load Views/Default.mat                      %get 3D view (def3D)
```

```
s.x1=1; s.x2=1; s.y1=0; s.y2=0; open=0;
[Norm Good] = getpath;                      %get the file path name
handles.value.FILE         = Good;
if ~isempty(varargin)                          %if the program is being opened by another program, wanting to open a
specific file, i.e. plotEv9('file',filename)
    handles.value.FILE     = varargin{end};  open = 1;
end
handles.value.sidegraphs.s = s;     handles.value.open         = open;
handles.value.rmin         = 1;     handles.value.rmax         = 200;
handles.value.emin         = 0;     handles.value.emax         = 4;
handles.value.bmin         = 0;     handles.value.bmax         = 4;
handles.value.e_points     = 0.05;  handles.value.calib_file   = '';
handles.value.delay1       = -0.5;  handles.value.delay2       = 3.5;
handles.value.e            = 0;     handles.value.camview      = def3D;
handles.value.eNe          = [];    handles.value.sidegraphs.s = s;
handles.value.linlog       = 750;   handles.value.graphs       = 10;
handles.value.scle         = 1;     handles.value.f            = [1 2 3 0];
handles.value.fit          = [];    handles.value.c            = [0.1 1 100000000 50;0 0 0 0];
handles.value.xofs         = 0;     handles.value.ub           = [0.1001 0.5001 100000000.0001 Inf];
handles.value.cc           = 0.18;  handles.value.lb           = [0.1 0.5 100000000 0];
handles.value.scldel       = 1;     handles.value.consec       = [1 1 1 1];
handles.value.bvals        = [];    handles.value.blankEv      = [0 300;0 1;8 10];
handles.value.ben          = 0;     handles.value.xyline       = [0.8 1];
handles.value.eimage       = 0;     handles.value.ebeta        = 0;
handles.value.eC           = [0;0.2;0.5];

guidata(gcf,handles)
updateui(handles)
set(handles.radon,  'Value' ,1);    radon_Callback(handles.radon,1,handles)
set(handles.intenon,'Value' ,1);    intenon_Callback(handles.radon,1,handles)


function updateui(handles) %update all the editbox values
handles=guidata(gcf);
set(handles.rmin,       'String',handles.value.rmin);
set(handles.rmax,       'String',handles.value.rmax);
set(handles.emin,       'String',handles.value.emin);
set(handles.emax,       'String',handles.value.emax);
set(handles.bmin,       'String',handles.value.emin);
set(handles.bmax,       'String',handles.value.emax);
set(handles.e_points,   'String',handles.value.e_points);
set(handles.delay1,     'String',num2str(handles.value.delay1,'%5.0f'));
set(handles.delay2,     'String',num2str(handles.value.delay2,'%5.0f'));
set(handles.displaytext,'String','');
set(handles.linlogedit, 'String',handles.value.linlog);
set(handles.ste,        'String',handles.value.rmin);
set(handles.stpe,       'String',handles.value.rmax);
set(handles.stdel,      'String',handles.value.delay1);
set(handles.stpdel,     'String',handles.value.delay2);
set(handles.xofs,       'String',handles.value.xofs);
set(handles.cc,         'String',handles.value.cc);
set(handles.scldel,     'String',handles.value.scldel);
set(handles.scle,       'String',handles.value.scle);
set(handles.t1,         'String',handles.value.c (1,1));
set(handles.t2,         'String',handles.value.c (1,2));
set(handles.t3,         'String',handles.value.c (1,3));
set(handles.t4,         'String',handles.value.c (1,4));
set(handles.t1fit,      'Value' ,handles.value.f(1)/1);
set(handles.t2fit,      'Value' ,handles.value.f(2)/2);
set(handles.t3fit,      'Value' ,handles.value.f(3)/3);
set(handles.t4fit,      'Value' ,handles.value.f(4)/4);
guidata(handles.t1,handles);

% Outputs from this function are returned to the command line.
function varargout = Analyse_Data_1_0_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
loadfile(eventdata, handles)
load MyColormaps/AnalyseDefault.mat
```

```
set(handles.figure1,'ColorMap',cmp)

%% Load Data Function.
function loadfile(eventdata, handles)
%eventdata=2 when viewing .dat files
global intdata beta rRad d
d=[];eon=0;b=[];
set(handles.displaytext,'String','Loading...');
FILE=handles.value.FILE;
if handles.value.open==0                %if you don't know the filename, open an "open file" window
    [name, pathstr] = uigetfile({'.mat','View Processed Data (*processed.mat)';'*processed.dat','View Saved Data
(*.dat)'},'Pick a TRPES file',[fileparts(FILE) '\*processed.mat']);
    if isequal(name, 0)                 %if the user presses "cancel" on the load file window
        set(handles.displaytext,'String',[sprintf('New File not loaded\nData File: ') FILE]), return
    else FILE=[pathstr name]; end       %save the new filename
end
[pathstr, name, ext] = fileparts(FILE);
subedit = [handles.bSub,handles.cSub,handles.bsmean,handles.csmean];
set(subedit,'Enable','off')
if   strcmp(get(handles.radon,'Enable'),'off'),set([handles.e_points,handles.radon],'Enable','on'),end%re-enable   the
commands.
set(handles.FitPanel,'Visible','off')

%.MAT FILE
if strcmp('.mat',ext)
    set([handles.betaon handles.fitdisp handles.beta4],'Enable','on')
    load(FILE);                                                                                    %   contains:
'delays','intdata','beta','rad','imdata','Isize','scantrace','scanrange')
    if exist('h','var')
        hh=handles.value;
        try
            handles.value=h;
            guidata(gcbo,handles)
            updateui(handles)
            plotscan(handles.slidEmax, 0, handles),
            return
        catch %if the program is changed in any way, this is likely to throw an error. the catch should rescue the
main data, although you will still lose all the other bits and bobs.
            handles.value = hh;
            delays   = h.delays;    beta = h.beta;
            intdata  = h.data;      rad  = h.rad;
            imdata   = h.image;     Isize= h.rmax;
            scanrange= h.scanrange;
        end
    end
    handles.value.scanrange = scanrange;
    handles.value.image     = imdata;
    handles.value.rmax      = Isize;
    handles.value.rad       = rad;
    handles.value.Re        = (1:size(intdata,2))*handles.value.rmax/size(intdata,2);
    handles.value.ftype     = 'mat';
%.DAT FILE
elseif strcmp('.dat',ext)               %If *.dat file is opened (viewing processed data files)
    %open file and extract values
    off=[handles.betaon,handles.fitdisp,handles.beta4,handles.graph_edit,handles.imprfit];
    set(off,'Enable','off')
    set(handles.intenon,'Value',1)
    fid = fopen(FILE,'r');f   = fscanf(fid,'%c');
    fclose(fid); p = findstr(f,'schritte');
    n = str2num(f(p+10:p+14)); p = findstr(f,'X-scale');  %get no. of delay positions
    r = f(p+9);      %find out if x-axis is in radius or energy
    p(1) = findstr(f,'Wellenlaengen')+18; p(2) = findstr(f,'XMode')-2; %find energy spacing
    e = str2num(f(p(1)+16:p(2)-1));             %get energy spacing
    p = findstr(f,'#')+1;                       %find main data
    l = findstr(f(p(3):end),sprintf('\n'))+p(3)-1;%get all the linebreaks (so you can just get the correct number of
lines)
    a = str2num(f(l(1):l(n+1)));                %extract main dataset
    p = findstr(f,'*')+1;                       %find background data (if it exists)
```

```
    for i=1:length(p)
        l = findstr(f(p(i):end),sprintf('\n'))+p(i)-1;   %get all the linebreaks (so you can just get the correct
number of lines)
        b(:,:,i) = str2num(f(l(3):l(n+4)));     %extract background dataset
    end
    %process and save extracted data
    if isempty(b)                               %if no background data
        set(subedit,'Value',0), bRad = []; cRad = [];
    else                                        %assign background data
        set(subedit,'Enable','on')
        bRad = b(:,2:end,1)';  bRad(isnan(bRad)) = 0;
        cRad = b(:,2:end,2)';  cRad(isnan(cRad)) = 0;
    end
    rRad   = a(:,2:end)';  rRad(isnan(rRad)) = 0;%assign main data
    delays = a(:,1);
    intdata= 1; es = sort([e(end) e(1)]);
    beta   = zeros(size(rRad,1),size(rRad,2),2);
    if strcmp(r,'t')                    %if x-axis is in radius
        handles.value.rmin  = es(1); handles.value.rmax = es(2);%save values
        handles.value.ftype = 'rdat';
    else                                %if x-axis is in energy
        handles.value.emin  = es(1); handles.value.emax  = es(2); %save values
        handles.value.EMAXX = es(2); handles.value.ftype ='edat';
        handles.value.e     = e;     handles.value.eNe   = rRad';
        set(handles.radon,'Enable','off'), set(handles.eon,'Value',1); eon=1;
        handles.value.calib_file='No calibration file required - viewing *.dat file';
    end
    handles.value.image     = zeros(480,480,length(delays));
    handles.value.rmax      = size(rRad,1);
    handles.value.rad       = rRad;
    handles.value.bRad      = bRad;
    handles.value.cRad      = cRad;
    handles.value.Re        = e;
end
intdata(isnan(intdata))=0;
if max(delays)<200; delays=delays*1000;end %work in fs
neg=1; if delays(end)<delays(1),neg=-1;end %if it's -ve, turn it +ve for now
handles.value.delays = delays; guidata(handles.slidEmax,handles)%required for linlogcalc
delmod = linlogcalc(handles,handles.value.linlog);%Check settings and make "delmod" (modified delay)

% SAVE VALUES
handles.value.data  = intdata;          handles.value.stpet = handles.value.rmax;
handles.value.beta  = beta;             handles.value.stet  = handles.value.rmin;
handles.value.open  = 0;                handles.value.stek  = handles.value.emin;
handles.value.stpdel= max(delmod);      handles.value.stpek = handles.value.emax;
handles.value.stdel = min(delmod);      handles.value.steb  = handles.value.emin;
handles.value.delmod= delmod;           handles.value.stpeb = handles.value.emax;
handles.value.FILE  = FILE;             handles.value.YTickL= [neg*delays(1),0,1000,10000,max(neg*delays)];
handles.value.delay2= max(neg*delays);  handles.value.delay1= min(neg*delays);
%update user interface
set(handles.linlogedit,  'String',handles.value.linlog);
% set(handles.displaytext, 'String',['Loaded: ' FILE]);
set(handles.figure1,'Name',['Analyse Data: ' FILE])
set(handles.delay1,      'String',num2str(handles.value.delay1,'%5.0f'));
set(handles.delay2,      'String',num2str(handles.value.delay2,'%5.0f'));
set(handles.ste,         'String',num2str(handles.value.stet));
set(handles.stpe,        'String',num2str(handles.value.stpet));
set(handles.emin,        'String',num2str(handles.value.emin));
set(handles.emax,        'String',num2str(handles.value.emax));
set(handles.stdel,       'String',num2str(handles.value.stdel));
set(handles.stpdel,      'String',num2str(handles.value.stpdel));
set(handles.rmax,        'String',num2str(handles.value.rmax));
set(handles.slidDmin,    'Min'   ,handles.value.delay1);
set(handles.slidDmin,    'Max'   ,handles.value.delay2);
set(handles.slidDmax,    'Min'   ,handles.value.delay1);
set(handles.slidDmax,    'Max'   ,handles.value.delay2);
set(handles.slidDmin,    'Value' ,handles.value.delay1);
set(handles.slidDmax,    'Value' ,handles.value.delay2);
```

```
if get(handles.radon,    'Value')
   set(handles.slidEmin, 'Min'   ,handles.value.rmin);
   set(handles.slidEmin, 'Max'   ,handles.value.rmax);
   set(handles.slidEmax, 'Min'   ,handles.value.rmin);
   set(handles.slidEmax, 'Max'   ,handles.value.rmax);
   set(handles.slidEmin, 'Value' ,handles.value.rmin);
   set(handles.slidEmax, 'Value' ,handles.value.rmax);
elseif get(handles.eon,  'Value')
   set(handles.slidEmin, 'Min'   ,handles.value.emin);
   set(handles.slidEmin, 'Max'   ,handles.value.emax);
   set(handles.slidEmax, 'Min'   ,handles.value.emin);
   set(handles.slidEmax, 'Max'   ,handles.value.emax);
   set(handles.slidEmin, 'Value' ,handles.value.emin);
   set(handles.slidEmax, 'Value' ,handles.value.emax);
end
guidata(handles.slidEmax,handles)
fclose('all')
if eon,eon_Callback(handles.eon,eventdata,guidata(handles.slidEmax))
else   plotscan(handles.slidEmax, 0, handles),end % plot data
%% Save File
function savefile(handles,saveas)
set(handles.displaytext,'String','Saving...');drawnow
file=handles.value.FILE;
h=handles.value;                    %#ok<NASGU>   get variables to save
if saveas
    [path  name  filter]=uiputfile({'*.mat','Full  Save  Format  (*.mat)';'*.dat','Export  Data  (*.dat)'},'Save  file
as',file);
    file=[name path];
    if filter==2,exportbtn_Callback(gcbo, file, handles),return,end
end
save(file,'h')
set(handles.displaytext,'String',['File Saved as ' file]);
% exportbtn_Callback(hObject, eventdata, handles)

%% 'See Amps' Button
function fittbutton_Callback(hObject, eventdata, handles)
try axes(findobj(get(handles.graphpanel,'Children'),'Tag','3Dplot')),end %make sure your looking at the right axis
%get the data
f     = handles.value.f;    [path name] = fileparts(handles.value.FILE(1:end-14));
f     = f(find(f));         xyline= handles.value.xyline;
c     = handles.value.c;    mx    = max(max(c(2:end,f)))*1.01;
normc = c/mx;               xl    = get(get(gca,'Xlabel'),'String');
e     = handles.value.x;    LS    = {'-';'--';':';'-.'};
IPline= [];                 fh    = figure;     % open a new figure
% expfig(fh,0,10,path,[name '_Amplitudes'],0)   % make it nice (using function above)
% co=get(gca,'ColorOrder');   set(gca,'ColorOrder',co(2:end,:))
dot=plot(e(1),normc(2,f(1)));hold all           %plot a dot to cycle the line color round one.#
for i=1:length(f)                               % for each fit
    hh(i)  = plot(e,normc(2:end,f(i)),'LineStyle',char(LS(i)));%plot the amplitude line
    leg(i) = cellstr(['\tau_' num2str(i) ' = ' num2str(c(1,f(i)),'%-5.2g') ' ps']); %create the legend title
end
hold off, axis tight, delete(dot), xax=xlim;
%set the axis limits (in case there is blank area on graph)
if strcmp(get(handles.blankEv,'Checked'),'on')
    if get(handles.bon,'Value'), xax=handles.value.ben-handles.value.blankEv(2,:);
    else xax=handles.value.blankEv(2,:);end
    if xax(1)>xax(2), xax=fliplr(xax);end
    xlim(xax)
end
% if 'Figure/Add line to 3D Graph' is on, add a dotted line to Amps graph
if xyline(2)==1 && strcmp(get(handles.addxyline,'Checked'),'on')
    if get(handles.bon,'Value'),xyline(1)=handles.value.ben-xyline(1);end
    IPline = line([xyline(1) xyline(1)],[0 0.26],'Color','black','LineStyle','--');
    hasbehavior(IPline,'legend',false)          % turn off it's legend entries for non important data
end
zeroline = line(xax,[0 0],'color','black');     % add a line at zero, to highlight negative dynamics
hasbehavior(zeroline,'legend',false)            % turn off it's legend entries for non important data
xlabel(xl), ylabel('Normalised Amplitudes'),legend(hh,leg,'Location','SouthOutside'),legend boxoff %annotate graph
```

```
pubfig(fh, [name '_Amplitudes'],path, 0.5)      % make plot pretty (using external function)

%% Plot Functions (functions called by main plot fn)
%get figure and graph handles (depends on whether user is exporting figure or not)
function [haxis haxisSS iaxis taxis saxis fh]=fighandles(eventdata,handles)
haxisSS = handles.MainGraph;
haxis   = handles.graphpanel;
iaxis   = handles.ImageGraph;
fh      = handles.figure1;
taxis   = handles.TimeSideGraph;
saxis   = handles.SpectraSideGraph;
if eventdata>=3 %setup exporting figure and handles to plot in
    mycmap = get(gcf,'Colormap');
    fh = figure('Visible','off');        %make new figure
    switch eventdata
        case 3,haxis = fh; haxisSS = gca; %main graph
        case {4,5},iaxis = gca;         %3D image
        case 6,taxis = gca;         %Time Graph
        case 7,saxis = gca;         %Spectra Graph
    end
    set(gcf,'Colormap',mycmap)      %if colormapeditor has been used, get the same colormap on new figure
end

%get lin/log axis lables
function [Nlog YTick YTickL] = linlogv(handles,irev,neg,xofs)
%get YTick Labels and make sure they are good (monitonically increasing)
YTickL = sort(handles.value.YTickL); %make sure they're in the right order
linlog = handles.value.linlog;
delays = handles.value.delmod - xofs;
if get(handles.linlogaxis,'Value') && linlog < delays(end) && length(delays)>5
    e      = 1:length(YTickL);
    for i=2:length(YTickL),if YTickL(i-1)==YTickL(i),e(i)=0;end,end %check that there are no duplicate values
    YTickL = YTickL(find(e)); YTick  = YTickL;%make a final, monitonically increasing, set of labels
    %get other values
    Nlog   = ceil(find1D(linlog,neg*delays));
    r      = floor(find1D(linlog,YTickL));
    %work out where YTick Labels should go on graph
    for i=r+1:length(YTick),m=round(find1D(YTickL(i),irev*(handles.value.delays-xofs)));YTick(i)=delays(m);end
    if irev==-1, YTick = fliplr(YTick); YTickL = fliplr(irev*YTickL);end
else Nlog=1; YTick=1; set(handles.linlogaxis,'Value',0)
end

%Do Background Subtraction (If Required)
function rad = backgroundsub(handles)
rad = handles.value.rad;
on  = ones(1,size(rad,2));
if get(handles.bSub,'Value') && strcmp(get(handles.bSub,'Enable'),'on')
    if get(handles.bsmean,'Value'),rad=rad-mean(handles.value.bRad,2)*on;
    else rad=rad-handles.value.bRad; end
end
if get(handles.cSub,'Value') && strcmp(get(handles.cSub,'Enable'),'on')
    if get(handles.csmean,'Value'), rad=rad-mean(handles.value.cRad,2)*on;
    else rad=rad-handles.value.cRad; end
end
if get(handles.tSub,'Value')
    rad=rad-mean(rad(:,1:2),2)*on;
end

%manage global fit if size of data to be fit has changed
function exit = updtatec(handles,zz,yy,fit)
exit=0;x=0;
%make c the right size
if (size(handles.value.c,1)-1)~=size(zz,2)
    handles.value.c(2:size(zz,2)+1,:) = 1; x=1;
    handles.value.c = handles.value.c(1:size(zz,2)+1,:);
end
%re-fit data
```

```
if get(handles.fiton,'Value')&&(any(size(fit)~= size(zz))|| x==1)%if you are on Global Fit mode, don't plot now
because 'fit' will be thr wrong size
    handles.value.z = zz; handles.value.y = yy;         %get the data values
    guidata(gcbo,handles),globfit(3)                    %save data and get new fit
    exit=1;                                             %exit plot fn (globfit calls plotscan with new fit to plot)
end


%get dataset & Fit Beta
function [points image beta e e1 e2 c z x calib on] = getdataset(handles,rad) %work out what kind of data you are
plotting
R       = handles.value.Re;
rmin    = handles.value.rmin;   emin    = str2double(get(handles.emin,'String'));
rmax    = handles.value.rmax;   emax    = str2double(get(handles.emax,'String'));
bmin    = str2double(get(handles.bmin,'String'));
bmax    = str2double(get(handles.bmax,'String'));
calib   = '';
c='Signal intensity vs '; z='Average intensity'; points=rad';

if get(handles.radon,'Value')       %RADIUS PLOT
    e  = R; d='Pump-Probe Delay vs Radius:'; x='Radius (pixels)';
    if e(1) > e(2), e=wrev(e);  points= fliplr(points);  end % sw=e1;e2=e1;e1=sw; %if energy is the wrong way round
    e1 = round(find1D(rmin,e));
    e2 = round(find1D(rmax,e));
    on = 1;
elseif get(handles.eon,'Value')     %KINETIC ENERGY PLOT
    d  = 'Pump-Probe delay vs Kinetic energy:'; x='Kinetic energy (eV)';
    [points e cal] = tof2ke(handles,points);
    [cpath cname cext] = fileparts(cal); calib = ['Calibration File: ' cname cext];
    e1 = round(find1D(emin,e));
    e2 = round(find1D(emax,e));
    on = 2;
else                                %BINDING ENERGY PLOT
    d  = 'Pump-Probe delay vs Binding energy:'; x='Binding energy (eV)';
    [points e cal] = tof2ke(handles,points);
    e  = handles.value.ben-e;
    if e(1) > e(2), e=wrev(e);  points= fliplr(points);  end % sw=e1;e2=e1;e1=sw; %if energy is the wrong way round
    [cpath cname cext] = fileparts(cal); calib = ['Calibration File: ' cname cext];
    e1 = ceil(find1D(bmin,e));
    e2 = ceil(find1D(bmax,e));
    on = 3;
end
% sort out image and BETA FIT
beta.gdata = double(handles.value.data);            %polar image
image      = double(handles.value.image);           %normal image
beta.gbeta = handles.value.beta;
beta.err   = 0;
if strcmp(get(handles.bfitype,'Checked'),'on'),ftype=1;else ftype=2;end
if get(handles.betaon,'Value')      %BETA PLOT
    % calculate the data
    c='Anisotropy parameter vs ';z='Anisotropy parameter  \beta_';      %(use \it\beta to make it italic)
    if on ~= 1                                                         %if energy binned
        if size(handles.value.eimage,2)~=length(e1:e2+1)                  %if the energy scaling has changed, or the
user has pressed 'Improve fit'
            beta.gdata = econvert(beta.gdata,cal,e,R);                 %re-bin the image into the energy scale
            erron = strcmp(get(handles.betaerron,'Checked'),'on');     %get robust errors
            try dat=beta.gdata(:,e1:e2+1,:);catch dat=beta.gdata(:,e1:e2,:);end
            [beta.gdata beta.gbeta] = IntegrateImage(dat,ftype,handles.value.ebeta,1,1,2,erron,''); %FIT BETA
            if all(all(all(beta.gdata==0)));
                beta.gdata = handles.value.eimage;  beta.gbeta = handles.value.ebeta;
            end
            handles.value.eimage = beta.gdata; handles.value.ebeta = beta.gbeta; %save the new image and beta fit
        else beta.gdata=handles.value.eimage; beta.gbeta = handles.value.ebeta; %use the values saved after the last
fit
        end
    end
    beta.gbeta(isnan(beta.gbeta))=0; p=points;
    % decide what data to show
```

```
    if get(handles.beta4,'Value'), par=3; z=[z '4']; else par=2; z=[z '2']; end
    points = beta.gbeta(:,:,par)'; if on==3, points = fliplr(points);end
    thresh = get(handles.BetaSlid,'Value');
    if size(beta.gbeta,3)>3; beta.err=beta.gbeta(:,:,par+2); %if there are error values
        th=(1-thresh)*mean(mean(beta.err));%get threshold value from beta slider (only visible in Anisotropy mode)
        if thresh ~= 0; points(beta.err'>th) = NaN; beta.err(beta.err>th) = NaN; end   %get rid of points below
threshold
    else
        beta.err=1/p;
        th=thresh*max(max(p))/2;%get threshold value from beta slider (only visible in Anisotropy mode)
        if thresh ~= 0; points(p<th) = NaN; end  %get rid of points below threshold
    end
end
if strcmp(get(handles.FitPanel,'Visible'),'on'),image=0;       %no image
elseif get(handles.polc,'Value')&&any(any(any(image))),image=beta.gdata; end%polar image
% if 1,mx=(max(max(max(image)))); image=exp(10)-exp((mx-image)*10/mx); end  %log scale image
guidata(handles.betaon,handles)                                  %save values
c=[c d];

%choose region of data to look at
function [idelay1 idelay2 stdel stpdel st1 stp1 st2 stp2]=choosedata(handles,e,e1,e2,irev,xofs)
esliders=[get(handles.slidEmin,'Value') get(handles.slidEmax,'Value')];
ste=esliders(1); stpe=esliders(2);
dsliders=[get(handles.slidDmin,'Value') get(handles.slidDmax,'Value')];
idelay1 = round(find1D(handles.value.delay1,irev*(handles.value.delays-xofs)));
idelay2 = round(find1D(handles.value.delay2,irev*(handles.value.delays-xofs)));
if idelay1 > idelay2; d=[idelay1 idelay2]; idelay1=d(2); idelay2=d(1); end
delays  = handles.value.delmod - xofs;
delay1  = delays(idelay1);      delay2  = delays(idelay2);
stdel   = handles.value.stdel;  stpdel  = handles.value.stpdel;
%work out where to start and stop scans
if stdel  < delay1 || stdel  > delay2, stdel = delay1; end
if stpdel < delay1 || stpdel > delay2, stpdel= delay2; end
%find out what data is selected for sidegraphs
if stpe < ste, sw=stpe; stpe=ste; ste=sw; end               %if st & stp are the wrong way round
if ste  < e(e1) || ste  > e(e2), ste  = e(e1); end          %if st is outside data
if stpe < e(e1) || stpe > e(e2), stpe = e(e2); end          %if stp is outside data
stp1=round(find1D(stpe,e)); st1=round(find1D(ste,e));       %find where they are in data
st2 =round(find1D(stdel,delays));stp2=round(find1D(stpdel,delays));
if get(handles.fitdisp,'Value')&&get(handles.betaon,'Value'),
    set(handles.slidDmax,'Enable','off'), stp2=st2;         %make both delays values the same
else
    set(handles.slidDmax,'Enable','on')
end
% calculate the slider steps and min/max positions
dstep = min(delays(2:end)-delays(1:end-1))/(delay2-delay1)*1.01;
if dstep>0.3,dstep=0.3;elseif ~isfinite(dstep),dstep=0.1; elseif dstep<=0,dstep=-dstep+0.001;end
if any(dsliders > delay2)||any(dsliders < delay1)
    set(handles.slidDmin,'Value',delay1);  set(handles.slidDmax,'Value',delay2);
end

%update UI
set(handles.ste,     'String', num2str(e(round(find1D(esliders(1),e))),'%5.2f'));
set(handles.stpe,    'String', num2str(e(round(find1D(esliders(2),e))),'%5.2f'));
set(handles.stdel,   'String', num2str(delays(st2),'%5.0f'));
set(handles.stpdel,  'String', num2str(delays(stp2),'%5.0f'));
set(handles.delay1,  'String', num2str(irev*handles.value.delays(idelay1)-xofs,'%5.0f'));
set(handles.delay2,  'String', num2str(irev*handles.value.delays(idelay2)-xofs,'%5.0f'));
set([handles.slidDmin, handles.slidDmax],'Min',delay1,'Max',delay2+0.001,'SliderStep',abs([dstep dstep*3]));

%calculate calibration constants and orange lines during calibration file creation
function [A E0 t0 eCscale]=calplot(handles,stp1,st1)
eC      = handles.value.eC;                         %get the energy spacing of molecule
A       = ((stp1^2-st1^2)/(max(eC)-min(eC)));       %calculate calibration values for scaling
E0      = min(eC);
t0      = st1;
eCscale = sqrt((eC - E0)*A + t0^2);                 %scale energy spacing to radius scale.
```

```
%create the waterfall plot
function hp=fillwaterfall(x,y,z)
%I ripped most of this code from the matlab waterfall.m function, but
%simplified it and changed it to make a filled in waterfall plot
[x,y]= meshgrid(x,y);                          %create x and y matrices
z0   = min(min(z));                            %decide how far down to fill in the slices
x    = [x(:,[1 1]) x x(:,size(x,2)*[1 1 1])];  %make the extra x and y points along the bottom
y    = [y(:,[1 1]) y y(:,size(y,2)*[1 1 1])];  %(so you have a 3D shape you can fill in)
z    = [z0*ones(size(x,1),1) z(:,1) z z(:,size(z,2)) z0*ones(size(x,1),2)]; %define the heights of those extra points
hp   = fill3(x',y',z',y');                     %plot the graph
set(hp,'LineWidth',2)
grid on                                        %show the background grid

% crosscorrelate data
function [odata odelays]=crosscdata(idata,idelays)
% odelays= idelays;
% odelays= idelays(find(idelays>=0))';
odelays=linspace(0,idelays(end),length(idelays));
odata  = zeros(length(odelays),size(idata,2));
data1  = resampledata(idata,idelays,odelays);
for i=1:length(odelays)
    data2      = resampledata(idata,idelays,odelays+odelays(i));
    odata(i,:) = trapz(odelays,data1.*data2);
end
odata=odata';
odata=(odata-mean(odata(:,end-2:end),2)*ones(1,size(odata,2)))';

% re-sample data (used in crosscorrelate data)
function odata=resampledata(idata,ix,ox)
%idata = data to be resampled; ix = original x axis spacing; ox = new x axis spacing
odata = zeros(length(ox),size(idata,2));
for i=1:length(ox)
    a = ceil (find1D(ox(i),ix)); b = floor(find1D(ox(i),ix));
    weight=(ox(i)-ix(b))/(ix(a)-ix(b));
    if a==b,weight =0.5; end
    odata(i,:) = (weight*idata(a,:)+(1-weight)*idata(b,:));
end

% weighted mean - returns the wighted mean values of the elements along
% different dimensions of an array using the errors of each element to
% calculate weights. Errors must be an array the same size as values.
function [wm err] = wmean(values, err, dim)
if nargin==2                        %find the best dimension (same as used in mean or sum)
    dim = min(find(size(x)~=1));    %#ok<MXFND>
    if isempty(dim), dim = 1; end
end
if length(err)~=1 && ~all(isequal(size(values),size(err)))
    error('Either the sizes of the values and errors must be the same or errors must be a single number.');
end
if length(err)==1;                %if a single value for error
    wm = sum(values,dim)/size(values,dim); %calculate a standard mean
else
    weights = 1./(err); weights(~isfinite(weights)) = 0;
    wm      = sum(values.*weights,dim)./sum(weights,dim);
    err     = sqrt(1./sum(weights,dim));
    for i=1:length(wm)
        a=i; b=1:size(values,dim);
        if dim==1, a=b; b=i; end
        err(i)              =             sqrt(1./sum(weights(a,b),dim).*(1./(size(values,dim)-1)).*sum(((values(a,b)-
wm(i)).^2.*weights(a,b)),dim));
    end
end
%% MAIN PLOT FUNCTION
function plotscan(hObject, eventdata, handles)
global Y
persistent figh
%eventdata=0 when called from most places
%eventdata=1 when called from camview
```

```
%eventdata=2 when calibration file creation mode is on
%eventdata=3 when called from exportfigure
% handles=guidata(handles.betaon);
if strcmp(get(handles.calibrategenerate,'Checked'),'on'),eventdata=2;end %if in process of creating calibration file
set(handles.displaytext,'String','Plotting...');
[pathstr, name]=fileparts(handles.value.FILE);
if isempty(name)
    loadfile(eventdata, handles),return %if a file hasn't been loaded, load one and exit plot fn
end
set([handles.SaveAs handles.SaveF],'Enable','on')
%define variables

try axes(findobj(get(handles.graphpanel,'Children'),'Tag','3Dplot')),end %make sure your looking at the right axis
% irev      = -2*get(handles.ccdata,'Value')+1; %-1 if ticked, +1 if not ticked
irev=1;
xofs    = handles.value.xofs;   mycmap  = colormap;
[az el] = view;                 emin    = str2double(get(handles.emin,'String'));
colour  = get(gcf,'Color');     emax    = str2double(get(handles.emax,'String'));
graphs  = handles.value.graphs; s       = handles.value.sidegraphs.s;
fit     = handles.value.fit;    delays  = handles.value.delmod-xofs;
if get(handles.fitresid,'Value'),fit    = handles.value.residual; end
se      = handles.value.scle;   sd      = handles.value.scldel;
meshx   = get(handles.meshx,'Value');meshy=get(handles.meshy,'Value');
if az~=0, handles.value.camview=[az el]; end                %save view
a=sprintf('\n\n');% fl=['Data File: ' handles.value.FILE sprintf('\n')];    %for display string
neg=1; if delays(end)<delays(1),neg=-1; end
s.x1=s.x1(:,2:end);s.y1=s.y1(:,2:end);s.x2=s.x2(2:end,:)';s.y2=s.y2(2:end,:)';%sidegraph data
load start
%if start && camm, handles.value.camview=camview;end%get camera view
if ~get(handles.fitdisp,'Value'),start=1;end
[pathstr, name] = fileparts(handles.value.FILE);

%go to plotting functions (above) to get get data etc
[Nlog YTick YTickL] = linlogv(handles,irev,neg,xofs);       %get lin/log plotting stuff
rad = backgroundsub(handles);                               %subtract background
[haxis haxisSS iaxis taxis saxis fh]  = fighandles(eventdata,handles);%work out where to plot everything
[points image beta e e1 e2 c z x calib on] = getdataset(handles,rad);%work out what data sets to plot
[idelay1 idelay2 stdel stpdel st1 stp1 st2 stp2] = choosedata(handles,e,e1,e2,irev,xofs);%work out where to start and
stop plotting the data
handles=guidata(handles.betaon);
%work out what data you are actually viewing
if ~get(handles.betaon,'Value'),er1=1; er2=1;
else, errs = beta.err';
    er1 = errs(idelay1:idelay2,st1:stp1);
    er2 = errs(st2:stp2,e1:e2);
end
xx = e(e1:e2); x2=xx;                                       %get values for x axis
yy = delays(idelay1:idelay2);                               %get values for y axis
zz = points(idelay1:idelay2,e1:e2);d1=[];d2=[];             %get 3D data for main graph
in = isnan(zz); zz(in)=0; points(isnan(points))=0;
yf = wmean(zz(:,st1-e1+1:stp1-e1+1),er1,2);                 %create time y values to fit to
% yf = mean(zz(:,st1-e1+1:stp1-e1+1),2);                    %create time y values to fit to
zl = [min(min(zz(~in))) max(max(zz(~in)))+0.0001];
handles.value.z = zz;                                       %save unaltered zz to fit to
if updtatec(handles,zz,yy,fit),return,end                   %if X Axis min, max or energy spacing has changed, get new
fit if required (go to fn above)
if get(handles.smoothZ,'Value'), for i=1:size(zz,2), zz(:,i)=smooth(zz(:,i),3);end, end %smooth data if required
if get(handles.fiton,'Value')                               %if global fit viewing
    zz = fit; d1 = yf;                                      %show fit data on main graph and get data points for delay
sidegraphs (plotted as 'o' points)
    d2 = wmean(points(st2:stp2,e1:e2),er2,1);                 %get data points for spectra sidegraphs (plotted as 'o'
points)
elseif get(handles.ccdata,'Value')
    [zz yy] = crosscdata(zz,yy);
    zl = [min(min(zz)) max(max(zz))];
    idelay1 = find(yy==0); if st2<idelay1,st2=idelay1;end
end
if get(handles.Zlog,'Value')                                %if viewing Z axis on Log Scale
```

```
    ZLmin = exp((1.1-get(handles.ZLSlid,'Value'))*5);
    zz(zz < zl(2)/ZLmin) = zl(2)/ZLmin; zz = log(zz);       %scale zz then log it
    if ~isempty(d1)
        points(points<zl(2)/ZLmin) = zl(2)/ZLmin; points = log(points); %scale points then log it
        d1 = wmean(points(idelay1:idelay2,st1-e1+1:stp1-e1+1),er1,2); %get new data points
        d2 = wmean(points(st2:stp2,e1:e2),er2,1);
    end
    zl = [min(min(zz(~in))) max(max(zz(~in)))];
end
% y1 = mean(zz(:,st1-e1+2:stp1-e1+1),2);    %get time data for side graph
[y1 er1] = wmean(zz(:,st1-e1+1:stp1-e1+1),er1,2);   %get time data for side graph
[y2 er2] = wmean(zz(st2-idelay1+1:stp2-idelay1+1,:),er2,1); %get spectra data for bottom graph
y2=y2'; er2=er2';
if get(handles.fiton,'Value') && get(handles.fitamps,'Value')
    ii=0;
    for i=find(handles.value.f),ii=1+ii;
        y1(:,end+1) = mean(Y(:,st1-e1+1:stp1-e1+1,ii),2);   %get fit of time data for side graph
        y2(:,end+1) = mean(Y(st2-idelay1+1:stp2-idelay1+1,:,ii),1); %get fit of spectra data for bottom graph
    end
end
if get(handles.normZ,'Value') && ~get(handles.betaon,'Value')  %normalise Z axis
    zz    = (zz    -zl(1))/(zl(2)-zl(1));
    points= (points-zl(1))/(zl(2)-zl(1));
    yf    = (yf    -zl(1))/(zl(2)-zl(1));
    d1    = (d1    -zl(1))/(zl(2)-zl(1));
    d2    = (d2    -zl(1))/(zl(2)-zl(1));
    y1    = (y1    -zl(1))/(zl(2)-zl(1));
    y2    = (y2    -zl(1))/(zl(2)-zl(1));
    zl    = [0 1];
end
xf = irev*handles.value.delays(idelay1:idelay2);            %create x values to fit (not lin/log or offset)
xl = yy;                                                    %create x values
zn = zz; zn(in)=nan;
if any(any(any(image)))
    if all(cell2mat(get([handles.fitdisp handles.polc handles.betaon],'Value')))
        image=mean(image(:,:,length(delays)+(st2:stp2)),3); %view the fit rather than the image
    else
        image=mean(image(:,:,(st2:stp2)),3);                %sum all the images between the slidebars
    end
end
if strcmp(get(handles.blankEv,'Checked'),'on')
    if get(handles.bon,'Value'),xax=handles.value.ben-handles.value.blankEv(2,:);
    else xax=handles.value.blankEv(on,:);end
else xax=[min(xx) max(xx)];end
if xax(1)==xax(2), return, end
if xax(1)>xax(2), xax=fliplr(xax);end
% zz=zz'; zz=(zz-mean(zz(:,end-2:end),2)*ones(1,size(zz,2)))';
%get data for plotting lines on graphs
if strcmp(get(handles.linlogshow,'Checked'),'on')
    ilog=Nlog-idelay1+1; ilog(ilog<1)=1;
    b=length(xx); range=round(b*0.8):b;
    if on==3, range=1:round(b*0.2); end
else ilog=1;
end
yons=ones(length(xx),1); xons=ones(length(yy),1);
xyline=handles.value.xyline;
ylg=(max(yy)-min(yy))/20;
xlg=(xax(2)-xax(1))/20;
if az<90 && az>-90, ytick=[min(yy) min(yy)-ylg];
else  ytick=[max(yy) max(yy)+ylg]; end
if az<0, xtick=[xax(1) xax(1)-xlg];
else  xtick=[xax(2) xax(2)+xlg]; end
cl='white'; ls='--'; xytick=0;
if strcmp(get(handles.xytick,'Checked'),'on'), cl = 'black'; ls=':'; xytick=1; end
if strcmp(get(handles.addxyline,'Checked'),'on')
    if xyline(2)==1
        if get(handles.bon,'Value'),xyline(1)=handles.value.ben-xyline(1);end
        if xytick, xln = [xyline(1) xyline(1)]; yln = ytick; zln = [zl(1) zl(1)];
```

```
        else, i = round(find1D(xyline(1),xx));
            xln = xx(i)*xons; yln = yy; zln = zz(:,i);
        end
    elseif xyline(2)==2
        if xytick, xln = xtick; yln = [xyline(1) xyline(1)]; zln = [zl(1) zl(1)];
        else, i = round(find1D(xyline(1),yy));
            xln = xx; yln = yy(i)*yons; zln = zz(i,:);
        end
    end
else xln = 0; yln = 0; zln = 0;
end
% if strcmp(get(handles.xytick,'Checked'),'on'),end

if get(handles.cam2D,'Value'),zln=zln*10;end

%PLOT GRAPHS
% Plot Side Graph 1(delay)
axes(taxis)
fit= tracefit(double(xf),double(yf),handles);   %fit decays according to 'Time Graph Fitting' listbox
yp=y1*sd; xp=x1; t=[];
if get(handles.betaon,'Value')      %plot points and errorbars rather than a continuous line
    if strcmp(get(handles.betaerron,'Checked'),'on')
%            errs = zeros(size(xp))+(mean(beta.err(st1:stp1,idelay1:idelay2)))*sd./sqrt(stp1-st1); %get the standard
deviation
    else er1 = std(zz(:,st1-e1+1:stp1-e1+1),0,2); end
    if length(errs)>2
        t = herrorbar(yp,xp,er1,'o'); set(t,'MarkerFaceColor','blue') %plot the errorbar plot (go to herrorbar fn
below)
        if isempty(s.y1(2:end,:)), yp  = []; xp = []; end   %dont plot the continous line unless the user has pressed
'capture'
    end
    %note: if you want the errorbars to show up even when user has pressed capture, use 'hold on' here
end
h = plot(yp,xp,s.y1(2:end,:)',s.x1(2:end,:)');          %plot delays graph
t(end+1:end+(length(h))) = h;                           %save handles
hold on
if ~isempty(fit), t(end+1) = plot(fit.y*sd,x1,'-r'); end%plot fit
if ~isempty(d1),  t(end+1) = plot(d1*sd,x1,'o');end     %plot data points (if applicable)
axis tight, xl=xlim; yl=ylim;                           %make the axis tight and get the limits
if xl(1)<0, t(end+1) = line([0 0],[x1(1)-1000 x1(end)*100],'color','black'); end %plot a black line at zero (so you
can see what is negative)
if get(handles.betaon,'Value'), if xl(1)<-1; xl(1)=-1;end, if xl(2)>2; xl(2)=2;end, end
xlim(xl)
if eventdata==6,                                        %if exporting plot
    t=t(t~=0); TX=get(t,'YData');TY=get(t,'XData');     %get the data on the graph
    ylabel(z); xlabel('Delay time (fs)');               %sort out formatting
    if length(t)==1,TX={TX};TY={TY};end
    for i=1:length(t),set(t(i),'XData',TX{i},'YData',TY{i}),end %flip it round so its not on it's side
    axis([yl xl])                                       %make it tight again
    if get(handles.linlogaxis, 'Value'),set(taxis,'XTick',YTick,'XTickLabel',YTickL);end %set the tick values for
linlog plot
    if     ilog~=1;plot([yy(ilog)     yy(ilog)],[xl(1)-0.1*(diff(xl))     xl(1)+0.1*(diff(xl))],'black','LineStyle','--
','Tag','Tck'),end%plot lin/log line
else
    set(taxis,'YTickLabel',''); xlabel(z); set(gca,'Color',colour);%format normal delays graph
end
hold off;
% Plot Bottom Graph 2(PE Spectra)
axes(saxis);
yp=y2*se;xp=x2;
if get(handles.betaon,'Value')      %plot points and errorbars rather than a continuous line
    if                                                  strcmp(get(handles.betaerron,'Checked'),'on'),%er2=
zeros(size(xp'))+(mean(beta.err(e1:e2,st2:stp2),2))*se./sqrt(idelay2-idelay1); %get the standard deviation
    else,er2 = std(zz(st2-idelay1+1:stp2-idelay1+1,:));end
    if length(er2)>2
        t    = errorbar(xp,yp,er2,'o'); set(t,'MarkerFaceColor','blue') %plot the errorbar plot (go to herrorbar fn
below)
```

```
        if isempty(s.y2(2:end,:)), yp  = []; xp = []; end   %dont plot the continous line unless the user has pressed
'capture'
    end
    %note: if you want the errorbars to show up even when user has pressed capture, use 'hold on' here
end
plot(xp,yp,s.x2(2:end,:),s.y2(2:end,:));hold on     %plot spectra graph
sc = abs(0.1*(x2(end)-x2(1)));
line([x2(1)-sc x2(end)+sc],[0 0],'color','black');      %plot a black line at zero (so you can see what is negative)
if ~isempty(d2),plot(x2,d2*se,'o'),end                  %plot data points (if applicable)
%plot bars if making calibration file
if eventdata==2
    [handles.cal.A handles.cal.E0 handles.cal.t0 eCscale] = calplot(handles,stp1,st1);
    stem(eCscale,[max(y2) max(y2) max(y2)],'Color','red')   %plot the scaled spacing on Spectra Graph
    a=sprintf(['A = ' num2str(handles.cal.A,'%5.0f') '\n\n Use the sliders to position the red bars on the peaks. Once
you are happy, click File/Calibration/Save Calibration File.\n\n']);
end, hold off, axis tight, xlim(xax)
if eventdata==7, xlabel(x)
else set(saxis,'XTickLabel',''); set(saxis,'Color',colour);%format normal spectra graph
end, ylabel(z);
% Plot imagegraph Graph (top left)
if any(any(any(image))) && ~get(handles.fiton,'Value')
    axes(iaxis),surf(image),caxis([0 1])                %plot image
    view(0,90),shading interp,axis image off            %make it look right
    if size(image,1)/size(image,2)>2,axis square,end    %if it's a long narrow image, make is square
end
% Plot Main Graph
axes(haxisSS)%plot main central plot
%plot 'display fits' beta graphs
if get(handles.fitdisp,'Value') && get(handles.betaon,'Value')%plot beta fits graphs
    %calculate variables
    gbeta = permute(beta.gbeta(st1:stp1,st2,:),[3 1 2]);
    fdata = beta.gdata(:,st1:stp1,length(delays)+st2);      %get the fit (it is stored at the end of the data)
    gdata = beta.gdata(:,st1:stp1,st2);                     %get the raw data
    xdata = (1:size(gdata,1))*180/size(gdata,1);            %create x values for plot
    slices= length(e(st1:stp1));
    if graphs>slices, graphs = slices;end                     %check that there are enough datapoints (at least 1 per
graph)
    g     = floor(slices/graphs);                             %devide data into sections of width g, one section for
each graph
    lines = ceil(graphs/6);                                   %if there are more than 6 graphs per column, make a new
column
    c     = ['Beta Parameter fit (green) and Data (blue) for time = ' num2str(stdel) ' ps'];
    figh  = subplotR(ceil(graphs/lines),lines,1,'Parent',haxis); %devide the space up into lots of small graphs
    for i=1:graphs %multiple graphs of anisotropy fit        %note, all graphs are taken from 1 datset 'st2', changed
by moving slider.
        [sss R] = max(gbeta(1,1+i*g-g:i*g));                %find the best slice R in current section (ignore sss)
        radd    = R+i*g-g;                                  %find where R is
        b       = gbeta(:,radd);                            %get beta peramiters for that R
        subplotR(ceil(graphs/lines),lines,i);               %make small graph that you will plot on current
        plot(xdata,gdata(:,radd),xdata,fdata(:,radd)),axis tight,grid off %plot raw data and beta fit for R
        title(texlabel(['e = ' num2str(e(st1+radd-1),3) ', beta_2 = ' num2str(b(2),3) ', beta_4 = ' num2str(b(3),3) ',
beta_0 = ' num2str(b(1),3)]))%create a title above each graph
        if meshx,set(axes_handle,'XGrid','on'),end          %make a mesh background if requested
        if meshy,set(axes_handle,'YGrid','on'),end
    end
    handles.value.betafitt=st1+radd-1;
%plot 'display fits' glabl fit graphs
elseif get(handles.fitdisp,'Value') && get(handles.fiton,'Value')
    ee=e(st1:stp1);graphs=length(ee);                       %make 1 graph per slice
    lines=ceil(graphs/5);                                     %if there are more than 6 graphs per column, make a new
column
    pt = points(idelay1:idelay2,e1:e2);
    figh  = subplotR(ceil(graphs/lines),lines,1,'Parent',haxis);%devide the space up into lots of small graphs
    for i=1:graphs %multiple graphs of global fit
        s=subplotR(ceil(graphs/lines),lines,i);             %make small graph that you will plot on current
        plot(x1,zz(:,i+st1-e1)),hold on,plot(x1,pt(:,i+st1-e1),'o'),hold off,axis tight %plot raw data and fit for
        if get(handles.linlogaxis, 'Value'), set(s,'XTick',YTick,'XTickLabel',YTickL);end %do lin/log tickmarks
        xlabel([num2str(ee(i)) ' eV'])%,legend('boxoff')           %create a title above each graph
```

```
        if meshx||meshy,grid on, end     %make a mesh background if requested
    end
%plot 3D coloured graph (normal view)
else
    %plot it
    subplotR(1,1,1,'Parent',haxis); figh = subplotR('Position',[0.1,0.09 0.87 0.87]); subplotR(figh);%do silly stuff
to make axis in the right place
    if get(handles.WaterGraph,'Value'),wf=1;h=fillwaterfall(x2,yy,zz);  %waterfall plot
    else wf=0; h=surf(x2,yy,double(zn));end          %normal plot
    set(figh,'Tag','3Dplot');
    %add extra bits
    axis tight,v=zlim;hold on
    if get(handles.Zlog,'Value'),zl(2) = v(end);
    else, zlim([zl(1) zl(2)]),end
    delete(findall(gcf,'Tag','Tck'))
    %plot white dotted lines at lin/log devision and user defined position
    if ilog~=1;
        if strcmp(get(handles.xytick,'Checked'),'on')
            xlog=xtick; ylog=[yy(ilog) yy(ilog)]; zlog=[zl(1) zl(1)];
        else xlog=xx(range); ylog=yons(range)*yy(ilog); zlog=zn(ilog,range); end%plot lin/log line
        line(xlog,ylog,zlog,'Linestyle',ls,'Color',cl,'Clipping','off','Tag','Tck')
    end
    line(xln,yln,zln,'Color',cl,'LineStyle',ls,'Clipping','off','Tag','Tck')%plot user added line
    %plot red lines showing where slider positions are
    if eventdata~=3                                  %don't plot the lines on the exported picture
        plot3([e(e1)   e(e2)  ],[delays(st2)     delays(st2)    ],[zl(2) zl(2)],'Color','red'),
        plot3([e(e1)   e(e2)  ],[delays(stp2)    delays(stp2)   ],[zl(2) zl(2)],'Color','red'),
        plot3([e(st1)  e(st1) ],[delays(idelay1) delays(idelay2)],[zl(2) zl(2)],'Color','red'),
        plot3([e(stp1) e(stp1)],[delays(idelay1) delays(idelay2)],[zl(2) zl(2)],'Color','red'),
        plot(handles.ColorBar,1)  ,colorbar(handles.ColorBar); %clear old colorbar & make new colorbar
    else                                             %format exported plot
        set(gca,'OuterPosition', [0 0 1 1]);
        if ~get(handles.cam3D,'Value'), colorbar('location','EastOutside');end
    end
    hold off, zt=0; xlim(xax)
    if get(handles.normZ,'Value'),ZTick=[0 0.5 1];ZTickL=['0.0';'0.5';'1.0'];zt=1;
    elseif get(handles.betaon,'Value'),ZTick=[-1,0,1,2];ZTickL=['-1';' 0';' 1';' 2'];zt=1;end
    if zt, set(figh,'ZTick',ZTick,'ZTickLabel',ZTickL),end %set the normalised data tick values
    if  get(handles.linlogaxis,  'Value'),set(figh,'YTick',YTick,'YTickLabel',YTickL);end  %set  the  tick  values  for
linlog plot
    %get the axis looking right
    if ~wf, shading interp; end
    if get(handles.cam3D,'Value'), view(handles.value.camview); rotate3d on,          %choose graph 3D position
    else rotate3d off, if wf, view(0,0), else view(0,90),end
    end
    xlabel(x), ylabel('Delay time (fs)')
    if ~wf,                                          %do 'X Mesh' and 'Y Mesh' stuff
        set(h,'EdgeColor','black')                   %make black lines in both directions
        if     meshx && ~meshy, set(h,'MeshStyle','Row')    %if only X Mesh, turn off Y lines
        elseif meshy && ~meshx, set(h,'MeshStyle','Column') %if only Y Mesh, turn off X lines
        elseif ~meshy&& ~meshx,set(h,'EdgeColor','none'),end%if neither, turn off all lines
    end
    lighting flat,set(gcf,'Renderer','Zbuffer')
end
% Reset the slider and section edit boxes, and save values,
set(handles.linlogedit,'String' ,handles.value.linlog);
set(handles.displaytext,'String',[calib a c]);%display text on UI
handles.value.YTickL        = YTickL; handles.value.y             = xf;
handles.value.sidegraphs.x1 = x1;     handles.value.sidegraphs.x2 = x2;
handles.value.sidegraphs.y1 = y1;     handles.value.sidegraphs.y2 = y2;
handles.value.srad          = rad;    handles.value.emax          = emax;
handles.value.x             = xx;     handles.value.emin          = emin;
set([handles.slidEmin, handles.slidEmax], 'Min', xax(1), 'Max', xax(2));
esliders=cell2mat(get([handles.slidEmin, handles.slidEmax],'Value'));
if any(esliders > xax(2)) || any(esliders < xax(1))
    set(handles.slidEmin,'Value',xax(1)); set(handles.slidEmax,'Value',xax(2));
end
```

```
save start start, guidata(hObject,handles)%save variables
%if exporting a figure:
figure(fh)
sz=0.55;    %sz = size (one digit = sqaure figure, 2 digits = [width height])
ud=1;       %ud = data that user can save from figure (ud=1 gets the data from the graph)
mt=[];      %mt = minor tickmarks ([xaxis yaxis zaxis]; or empty means no minor tickmarks)
FS=15;      %font size
switch eventdata
    case 3,n = '3D';     sz = [0.7 0.5];mt = [1 0 1];    %Main Graph;
           A =[0;x1(:)]; ud = [x2;zz];  ud = [A ud];     %construct a table with the time and energy values along each
side
           tick('X',0.1)
    case 4,n = 'Image';  ud = image;                    %2D Image
    case 5,n = 'Image';  ud = image; axis(iaxis), view(3), rotate3d on
    case 6,n = 'Time';   mt = [0 3 0]; FS = 17; tick('Y',0.1) %Time Graph
    case 7,n = 'Spectra';mt = 1;                        %Spectra Graph
end
if eventdata>=3 %if you are exporting a figure
    pubfig(fh, [name(1:end-9) n], pathstr, sz, ud, FS, 2, mt,3,0) %go to external pubfig function (see help pubfig for
definition of input variables)
    if eventdata==6 && get(handles.betaon,'Value'),xlim([-60 2988]),end
    if eventdata==3, set(fh,'PaperOrientation','landscape'),end
end
set(fh,'Visible','on')
close(findall(0,'-property','Alphamap','Visible','off'))%close any stray hidden figures (caused by crashes etc)

%set default axis tickmark spacing
function tick(axis,spacing)
switch axis
    case 'Y',  lim = ylim;
    case 'Z',  lim = zlim;
    otherwise, lim = xlim; axis = 'X';
end
tik = floor(lim(1)/spacing)*spacing: spacing :floor(lim(2)/spacing)*spacing; %default to 'spacing' devision tickmarks
if length(tik)<8 && length(tik)>2, %but only if it makes sense
    set(gca, [axis 'Tick'], tik)
end
%%   Export Data
%exporting data into Helmut's program format
function exportbtn_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
delaysN = length(handles.value.delays); %points per scan
display('opening export file...')
%reverse axis if required
delays=(handles.value.delays-handles.value.xofs)/1000;
n1=delaysN;%THIS IS WRONG! FIND OUT WHAT n1 IS.
subprobe = get(handles.cSub,'Value'); subpump = get(handles.bSub,'Value');
data=handles.value.srad';
[pathstr, name] = fileparts(handles.value.FILE);
if ischar(eventdata),[PathName,FileName]=fileparts(eventdata);
else [FileName,PathName] = uiputfile('*.dat','Save file as',[pathstr '/' name(1:end-13)]);
end

if FileName, fid = fopen( fullfile(PathName,FileName) , 'wt');
    display('writing data...')
    fprintf(fid, '# TRPES-data, exported into Helmut''s format from PlotE \n#\n');
    if get(handles.eon,'Value') %Kinetic Energy (e) radio button pressed
        [eNe e cal new] = tof2ke(handles,data);
        if new, handles.value.EMAXX=max(e); guidata(hObject,handles),end
        fprintf(fid,       'Kommentar       =       pumPOnly_substracted=%d;       probEOnly_substracted=%d;       X-
scale=%s;\n',subpump,subprobe,'e');
        fprintf(fid, 'schritte = %d \n', delaysN); %or "n1+n2"
        fprintf(fid, 'lschritte = %d \n', n1);
        fprintf(fid, 'Kanalanzahl = %d \n', length(e));
        fprintf(fid, 'data_type = TRPES \n');
        fprintf(fid, 'Wellenlaengen = '); fprintf(fid, '%6.5f ', e'); fprintf(fid, '\n');
        fprintf(fid, 'XMode = SingleX \n');
```

```
        fprintf(fid, 'Data = \n#\n');
        for i=1:delaysN
            %Every line starts with delay in picosec which is followed by corresponding PES
            ab = sprintf('%6.5f ', delays(i));
            bc = sprintf('%6.5f ', eNe(i,:));
            fprintf(fid,[ab bc '\n']);
        end;

    elseif get(handles.radon,'Value') %Radius radio button pressed
        Re = handles.value.Re;
        fprintf(fid,       'Kommentar       =       pumPOnly_substracted=%d;       probEOnly_substracted=%d;       X-
scale=%s;\n',subpump,subprobe,'et');
        fprintf(fid, 'schritte = %d \n', delaysN); %or "n1+n2"
        fprintf(fid, 'lschritte = %d \n', n1);
        fprintf(fid, 'Kanalanzahl = %d \n', handles.value.rmax-handles.value.rmin);
        fprintf(fid, 'data_type = TRPES \n');
        fprintf(fid, 'Wellenlaengen = ');
        fprintf(fid, '%6.1f ', Re(1,handles.value.rmin:handles.value.rmax));
        fprintf(fid, '\n');
        fprintf(fid, 'XMode = SingleX \n');
        fprintf(fid, 'Data = \n#\n');
        for i=1:delaysN
            %Every line starts with delay in picosec which is followed by corresponding PES
            fprintf(fid, '%6.4f ', delays(i));
            fprintf(fid, '%6.4f ', data(i,handles.value.rmin:handles.value.rmax));
            fprintf(fid, '\n');
        end;
    end
    fclose(fid);
    disp=['Data exported to: ' fullfile(PathName,FileName)];
else
    disp='Not exported this time.';
end
set(handles.displaytext,'String',disp)
display(disp)

%% Calibration Files
% Change Calibration File
function calibratechange_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
handles.value.calib_file='';%reset values
handles.value.eNe='';
guidata(hObject,handles)%seve reset values
set(handles.eon,'Value',1)%pretend user has clicked on 'Kinetic Energy' Radio Button
eon_Callback(hObject, eventdata, handles)

% Create Calibration File
function calibrategenerate_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
disp='No Calibration File Created This Time';
answer=questdlg('Is the Data currently loaded a recent spectra of a well characterised molecule?');
if strcmp('Yes',answer)     %if user presses Yes
    %create input dialog
    titles    =  {sprintf('Enter  Known  Energies:\n\nPeak  1  Energy  (eV):'),'Peak  2  Energy  (eV):','Peak  3  Energy
(eV):'};%titles for editboxes
    vals    = inputdlg(titles,'Enter Energies',1,cellstr(num2str(handles.value.eC)));
    if ~isempty(vals)
        disp='Use  the  top  sliders  to  position  the  bars  over  the  peaks.  Once  you  are  finished,  choose  "File/Save
Calibration File".';
        handles.value.eC = str2num(char(vals));
        set(handles.displaytext,'String',disp)
        set(hObject,            'Checked','on')
        set(handles.calibrategenexit,'Enable', 'on')
        guidata(hObject,handles)
        plotscan(hObject, 0, handles)
    end

elseif strcmp('No',answer)  %if user presses No
```

```
    disp='Please open a recent spectra of a well characterised molecule like butadiene or nitric oxide, then try
again.';
end                         %if user presses Cancel then don't do anything
helpdlg(disp)
clear

% Create Calibration File
function calibrategenexit_Callback(hObject, eventdata, handles)
handles=guidata(hObject);hObject=handles.calibrategenexit;
disp='No Calibration file saved this time';
[path,name] = fileparts(handles.value.FILE);
[name,path] = uiputfile('*.cal','Save file as',[path '\' name(1:end-13) '.cal']);
if name                                         %if the user doesn't press cancel
    fid = fopen(fullfile(path,name) , 'wt');        %open the file
    fprintf(fid,['A='  num2str(handles.cal.A ) '\n']);
    fprintf(fid,['E0=' num2str(handles.cal.E0) '\n']);
    fprintf(fid,['t0=' num2str(handles.cal.t0) '\n']);
    fclose(fid);                                    %close the file
    disp=['Calibration File Created: ' name];
end
set(hObject,'Enable', 'off')
set(handles.calibrategenerate,'Checked', 'off')
plotscan(hObject, 0, handles)
helpdlg(disp)

%% X-AXIS Sectionbox

%RADIUS
%Radius Radio Button
function radon_Callback(hObject, eventdata, handles)
on  =[handles.rmax,handles.rmin];
off =[handles.emax,handles.emin,handles.e_points,handles.etext,handles.bmax,handles.bmin];
if get(handles.radon,'Value');
    set(off,'Enable' ,'off'), set(on,'Enable' ,'on')
    if eventdata==1, else plotscan(hObject, 0, handles), end
end

%Radius Max
function    rmax_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function rmax_Callback(hObject, eventdata, handles)
rmax = str2double(get(hObject, 'String'));%get the new rmax value
if isnan(rmax)
    set(hObject, 'String', handles.value.rmax);
    errordlg('Input must be a number','Error');
else
    handles.value.rmax = rmax; guidata(hObject,handles)%Save rmax value
    plotscan(hObject, 0, handles)
end

%Radius Min
function    rmin_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function rmin_Callback(hObject, eventdata, handles)
rmin = str2double(get(hObject, 'String'));%get the new rmin value
if isnan(rmin)
    set(hObject, 'String', handles.value.rmin);
    errordlg('Input must be a number','Error');
else
    handles.value.rmin = rmin; guidata(hObject,handles)%Save rmin value
    plotscan(hObject, 0, handles)
end

% KINETIC ENERGY
% Kinetic Energy Radio Button
function eon_Callback(hObject, eventdata, handles)
off=[handles.rmax,handles.rmin,handles.bmax,handles.bmin];
on =[handles.emax,handles.emin,handles.e_points,handles.etext];
```

```
if get(handles.eon,'Value')
    if get(handles.cam3D,'Value') && strcmp(get(handles.rmax,'Enable'),'off')
        [az el]=view; handles.value.camview=view([-az el]); view(2),
    end
    set(off,'Enable' ,'off'), set(on,'Enable' ,'on')
    if isempty(handles.value.eNe)%if first time
        [eNe e cal]   = tof2ke(handles,handles.value.rad');
        handles.value.eNe = eNe;
        handles.value.e   = e;
        handles.value.calib_file = cal;
        handles.value.EMAXX= str2double(get(handles.emax,'String'));
        guidata(hObject,handles)
    end
    plotscan(hObject, 0, handles)
end
get(handles.cam3D,'Value')

% Kinetic Energy Max
function    emax_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function emax_Callback(hObject, eventdata, handles)
emax = str2double(get(hObject, 'String'));%get the new emax value
if isnan(emax)||emax<handles.value.emin
    set(hObject, 'String', handles.value.emax);%put back to orig. value
    errordlg(['Energy    Maximum    must    be    a    positive    number    between    Energy    Minimum    and    '
num2str(handles.value.EMAXX)],'Error');
else
    if emax>handles.value.EMAXX, set(hObject, 'String', handles.value.EMAXX);end
    handles.value.emax  = emax;
    handles.value.stpek = emax;
    guidata(hObject,handles) %Save emax value
    plotscan(hObject, 0, handles)
end

% Kinetic Energy Min
function    emin_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function emin_Callback(hObject, eventdata, handles)
emin = str2double(get(hObject, 'String'));%get the new emin value
if isnan(emin)||emin<0||emin>handles.value.emax
    set(hObject, 'String', handles.value.emin);%put back to orig. value
    errordlg('Energy Minimum must be a positive number less than Energy Maximum','Error');
else
    handles.value.emin = emin;
    handles.value.stek = emin;
    guidata(hObject,handles)%Save emin value
    plotscan(hObject, 0, handles)
end

% Kinetic Energy resolution
function   e_points_CreateFcn(hObject,   eventdata,   handles),if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function e_points_Callback(hObject, eventdata, handles)
e_points = str2double(get(hObject, 'String'));      %get the new e_points value
if isnan(e_points)
    set(hObject, 'String', handles.value.e_points); %put back to orig. value
    errordlg('Input must be a number','Error');
else
    handles.value.e_points = e_points;              %Save e_points value
    guidata(hObject,handles), plotscan(hObject, 0, handles)
end

%BINDING ENERGY
function bon_Callback(hObject, eventdata, handles)
off=[handles.rmax,handles.rmin,handles.emax,handles.emin];
on =[handles.bmax,handles.bmin,handles.e_points,handles.etext];
if get(handles.bon,'Value')
    if get(handles.cam3D,'Value'),[az el]=view; handles.value.camview=view([-az el]); view(2), end
```

```
    set(off,'Enable' ,'off'), set(on,'Enable' ,'on')
    if isempty(handles.value.bvals),bset_Callback(hObject, eventdata, handles),end
    if isempty(handles.value.eNe)%if first time
        handles           = guidata(hObject);
        [eNe e cal]       = tof2ke(handles,handles.value.rad');
        handles.value.eNe = eNe;
        handles.value.e   = e;
        handles.value.calib_file = cal;
        handles.value.EMAXX= str2double(get(handles.emax,'String'));
        guidata(hObject,handles)
    else, plotscan(hObject, 0, handles)
    end
end

function    bmin_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function bmin_Callback(hObject, eventdata, handles),plotscan(hObject, 0, handles)

function    bmax_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function bmax_Callback(hObject, eventdata, handles),plotscan(hObject, 0, handles)

%Change E button
function bset_Callback(hObject, eventdata, handles)
if isempty(handles.value.bvals),defans={'267','300'};
else defans = cellstr(num2str(handles.value.bvals')); end
answer = inputdlg({'Pump Wavelength (nm):','Probe Wavelength (nm):'},'',1,defans);
if isempty(answer),set(handles.eon,'Value',1)
    eon_Callback(hObject, eventdata, handles)
else
    for i=1:2,a(i)=str2double(char(answer(i))); end
    if any(isnan(a))||any(a<=0)
        errordlg('Please enter positive numbers only'),uiwait,
        bset_Callback(hObject, eventdata, handles)
    else
        handles.value.bvals= a;
        handles.value.ben  = 1240/a(1)+1240/a(2);
        e=handles.value.ben-handles.value.e;
        handles.value.steb = e(end);
        handles.value.stpeb= e(1);
        set(handles.bmin,'String',num2str(handles.value.steb ,3))
        set(handles.bmax,'String',num2str(handles.value.stpeb,3))
        guidata(hObject,handles)
        plotscan(hObject, 0, handles)
    end
end
%% Y AXIS(Delay Time) Sectionbox
% Y axis Max
function    delay2_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function delay2_Callback(hObject, eventdata, handles)
delay2 = str2double(get(hObject, 'String'));%get the new delay2 value
if isnan(delay2)
    set(hObject, 'String', handles.value.delay2);%put back to orig. value
    errordlg('Input must be a number','Error');
else
    handles.value.delay2 = delay2;
    if subplus(delay2), if delay2 < handles.value.stpdel, handles.value.stpdel=delay2; end
    elseif delay2 > handles.value.stpdel, handles.value.stpdel=delay2; end
    guidata(hObject,handles)%Save delay2 value
    plotscan(hObject, 0,  handles)
end

% Y axis Min
function    delay1_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function delay1_Callback(hObject, eventdata, handles)
delay1 = str2double(get(hObject, 'String'));%get the new delay1 value
```

```
if isnan(delay1)
    set(hObject, 'String', handles.value.delay1);%put back to orig. value
    errordlg('Input must be a number','Error');
else
    handles.value.delay1 = delay1;
    if subplus(delay1), if delay1 < handles.value.stdel, handles.value.stdel=delay1; end
    elseif delay1 > handles.value.stdel, handles.value.stdel=delay1; end
    guidata(hObject,handles)%Save delay1 value
    plotscan(hObject, 0,  handles)
end

% Cross Correlate Data Tickbox
function ccdata_Callback(hObject, eventdata, handles),plotscan(hObject, 0,handles)

%x offset
function    xofs_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function xofs_Callback(hObject, eventdata, handles)
global d,d=[];
value=fitedt(hObject,handles.value.xofs); %go to fitedt function to get value
if ~isempty(value),  handles.value.xofs = value; guidata(hObject,handles),
    if get(handles.fiton,'Value'),fitscan(handles)
    else,plotscan(hObject, 0,handles),end
end

% Lin/Log Checkbox
function linlogaxis_Callback(hObject, eventdata, handles)
handles.value.delmod = linlogcalc(handles,handles.value.linlog); %save modified delays vector
guidata(hObject,handles)
plotscan(hObject,0,handles)

% Lin/Log Edit box
function   linlogedit_CreateFcn(hObject,   eventdata,   handles),if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function linlogedit_Callback(hObject, eventdata, handles)
%get parameters
linlog = str2double(get(hObject, 'String'));%get the new linlog value
handles.value.delmod = linlogcalc(handles,linlog); %save modified delays vector
handles.value.linlog = linlog; set(handles.slidDmin,'Value',linlog)
guidata(hObject,handles)
plotscan(hObject,0,handles)

%calculates lin/log y scale
function delays=linlogcalc(handles,linlog)
if handles.value.delays(end)<handles.value.delays(1),neg=-1;else neg=1;end,
delays = neg*handles.value.delays/1000;       %turn it to +ve ps for now
if get(handles.linlogaxis,'Value'),
    Nlog   = round(find1D(linlog/1000,delays));     %find where linlog is in "delays" vector
    Nlogv  = delays(Nlog)-log10(delays(Nlog));     %find out the starting value
    delays(Nlog:end)=Nlogv+log10(delays(Nlog:end)); %calculate new vector
end
delays = neg*delays*1000;              %return delays to fs and to original rotation
%NOTE if 'linlogaxis' tickbox is not ticked, this function returns the
%original delays vector unmodified

%% FIGURE Sectionbox
% Camara modes
function cam2D_Callback(hObject, eventdata, handles),plotscan(hObject,1, handles)
function cam3D_Callback(hObject, eventdata, handles),plotscan(hObject,1, handles)
function WaterGraph_Callback(hObject, eventdata, handles),plotscan(hObject,0, handles)
function meshx_Callback(hObject, eventdata, handles)
if get(hObject,'Value'),set(handles.WaterGraph,'Value',0),end,plotscan(hObject,0, handles)
function meshy_Callback(hObject, eventdata, handles),plotscan(hObject,0, handles)
if get(hObject,'Value'),set(handles.WaterGraph,'Value',0),end,plotscan(hObject,0, handles)

%% SUBTRACTION Sectionbox
function tSub_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)%Subtract t<0 Tickbox
function bSub_Callback(hObject, eventdata, handles),set(handles.tSub,'Value',0) %Subtract Pump Tickbox
```

```
if get(hObject,'Value'),a='on';else,a='off';end, set(handles.bsmean','Enable',a),plotscan(hObject,0,handles)
function cSub_Callback(hObject, eventdata, handles),set(handles.tSub,'Value',0) %Subtract Probe Tickbox
if get(hObject,'Value'),a='on';else,a='off';end, set(handles.csmean','Enable',a),plotscan(hObject,0,handles)
function bsmean_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)%Subtract Mean Pump Tickbox
function csmean_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)%Subtract Mean Probe Tickbox

%% Z AXIS Sectionbox
% Intensity
function intenon_Callback(hObject, eventdata, handles)
off=[handles.beta4,handles.BetaMenu,handles.graph_edit,handles.imprfit,handles.fitdisp,handles.fitresid];
on =[handles.slidDmax,handles.ccdata];
set(on,'Enable','on'),set(off,'Enable','off')
set([handles.FitPanel handles.BetaSlid],'Visible','off')
if eventdata==1,else,plotscan(hObject,0,handles),end
% Anisotropy
function betaon_Callback(hObject, eventdata, handles)
off=[handles.graph_edit,handles.fitresid];
on =[handles.imprfit,handles.beta4,handles.BetaMenu,handles.slidDmax,handles.fitdisp];
set(on,'Enable','on'),set(off,'Enable','off')
set(handles.FitPanel,'Visible','off')
set(handles.BetaSlid,'Visible','on')
plotscan(hObject, 0, handles)
% Global Fit
function fiton_Callback(hObject, eventdata, handles)
off=[handles.beta4,handles.BetaMenu,handles.graph_edit,handles.ccdata];
on =[handles.slidDmax,handles.fitresid,handles.fitdisp,handles.imprfit];
set(on,'Enable','on'),set(off,'Enable','off')
set(handles.BetaSlid,'Visible','off'),set(handles.FitPanel,'Visible','on')
plot(handles.ImageGraph,0), axis(handles.ImageGraph,'off')
setupoptplot(handles)
plotscan(hObject, 0, handles)

%log scale tickbox
function Zlog_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)
if get(hObject,'Value'),set(handles.ZLSlid,'Visible','on')
else, set(handles.ZLSlid,'Visible','off'), end
%beta4 tickbox
function beta4_Callback(hObject, eventdata, handles),   plotscan(hObject,0,handles)
% Residuals tickbox
function fitresid_Callback(hObject, eventdata, handles),plotscan(hObject,0,handles)
%normalise Tickbox
function normZ_Callback(hObject, eventdata, handles),   plotscan(hObject,0, handles)
%smooth tickbox
function smoothZ_Callback(hObject, eventdata, handles), plotscan(hObject,0, handles)
%Log Slidebar
function      ZLSlid_CreateFcn(hObject,      eventdata,      handles),if      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function ZLSlid_Callback(hObject, eventdata, handles),  plotscan(hObject,0,handles)
%Beta Slider
function      BetaSlid_CreateFcn(hObject,      eventdata,      handles),if      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function BetaSlid_Callback(hObject, eventdata, handles),plotscan(hObject,0,handles)

% Display fits Tickbox
function fitdisp_Callback(hObject, eventdata, handles)
load start
off=[handles.slidDmax,handles.beta4];
on=[handles.graph_edit];
if get(hObject,'Value'),set(off,'Enable','off'),set(handles.imprfit,'Enable','on')
else set(on,'Enable','off'),set(off,'Enable','on'),end
if get(handles.betaon,'Value')&& get(hObject,'Value'),set(on,'Enable','on'),end
plotscan(hObject,0, handles)
start=0; save start start;

% Improve fit button
function imprfit_Callback(hObject, eventdata, handles)
if     get(handles.betaon,'Value'),handles.value.eimage=[1 0 1];guidata(hObject,handles),plotscan(hObject,0,handles)
elseif get(handles.fiton, 'Value'),globfit(3),end
```

```
%% FILE Menu
function FileMenu_Callback(hObject, eventdata, handles)
function SaveF_Callback(hObject, eventdata, handles),savefile(handles,0)
function SaveAs_Callback(hObject, eventdata, handles),savefile(handles,1)
function loadfile_Callback(hObject, eventdata, handles),loadfile(eventdata,handles)
function addfile_Callback(hObject, eventdata, handles),addfile('file',handles.value.FILE) %go to seperate 'addfile'
program
function GenData_Callback(hObject, eventdata, handles),ImageGenerator

function ProcessData_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
cfile = [handles.value.FILE(1:end-13) 'settings.mat'];
try close Process_Data, end  %If Process Data is open, close it
Process_Data_1_0('file',cfile)   %Open Process Data with current file (if valid)
%uistack(gcf,'bottom')       %sent PlotE backwards so Process_Data is on top.

%% FIGURE Menu
function FigureMenu_Callback(hObject, eventdata, handles)
function ExpMain_Callback(hObject, eventdata, handles), plotscan(hObject,3,handles)%main graph
function ExpIm_Callback(hObject, eventdata, handles),   plotscan(hObject,4,handles)%image
function ExpIm3D_Callback(hObject, eventdata, handles), plotscan(hObject,5,handles)%3D image
function ExpDel_Callback(hObject, eventdata, handles),  plotscan(hObject,6,handles)%Time Graph
function ExpSpec_Callback(hObject, eventdata, handles), plotscan(hObject,7,handles)%Spectra Graph
function eddels_Callback(hObject, eventdata, handles),  deledit(handles.value.FILE)%Edit delays
function cmpSave_Callback(hObject, eventdata, handles), cmp=get(gcbf,'ColorMap'); uisave('cmp','MyColormaps/*.mat')
%#ok<NASGU>
function      cmpLoad_Callback(hObject,      eventdata,      handles),      uiopen('MyColormaps/AnalyseDefault.mat'),
set(gcbf,'ColorMap',cmp)

%Save 3D View
function def3D_Callback(hObject, eventdata, handles),
def3D = view;                                                                    %#ok<NASGU>
if get(handles.bon,'Value'),[az el] = view; def3D = view(-az,el); view(az,el), end  %#ok<NASGU> if binding energy,
flip the view
uisave('def3D','Views/*.mat')

%Use Default 3D View
function usedef3D_Callback(hObject, eventdata, handles),
load('Views/Default.mat')
if get(handles.bon,'Value'),[az el] = view(def3D); view([-az el]); %if binding energy, flip the view
else view(def3D), end

%Load 3D View
function loaddef3D_Callback(hObject, eventdata, handles)
uiopen('Views/Default.mat')
if get(handles.bon,'Value'),[az el] = view(def3D); view([-az el]); %if binding energy, flip the view
else view(def3D), end

% Robust beta error
function betaerron_Callback(hObject, eventdata, handles)
if strcmp(get(hObject,'Checked'),'on')
     set(hObject,'Checked','off'),set(handles.betaerroff,'Checked','on')
else,set(hObject,'Checked','on'), set(handles.betaerroff,'Checked','off')
end
handles.value.eimage=0; plotscan(hObject,0,handles)

% Quick beta error
function betaerroff_Callback(hObject, eventdata, handles)
if strcmp(get(hObject,'Checked'),'on')
     set(hObject,'Checked','off'),set(handles.betaerron,'Checked','on')
else,set(hObject,'Checked','on'), set(handles.betaerron,'Checked','off')
end
handles.value.eimage=0; plotscan(hObject,0,handles)

% Don't Fit Beta 4
function bfitype_Callback(hObject, eventdata, handles)
if strcmp(get(hObject,'Checked'),'on')
```

```
     set(hObject,'Checked','off'),set(handles.beta4,'Enable','on')
else,set(hObject,'Checked','on'), set(handles.beta4,'Enable','off','Value',0)
end
handles.value.eimage=0; plotscan(hObject,0,handles)

%Edit Lin/Log Tickmarks
function YTick_Callback(hObject, eventdata, handles)
string='Please Enter a list of Time Delays you would like to see labelled';
YTick = inputdlg(string,'Lin/Log Tickmarks',1,{num2str(handles.value.YTickL)});
if ~isempty(YTick)
    YTick = eval(['[' char(YTick) ']']);
    if any(isnan(YTick)),errordlg('Please choose a list of numbers')
    else handles.value.YTickL=YTick;guidata(hObject,handles),end
    plotscan(hObject, 0, handles)
end

%Add line to 3D Graph
function addxyline_Callback(hObject, eventdata, handles)
if strcmp(get(hObject,'Checked'),'on')
    set(hObject,'Checked','off'), plotscan(hObject, 0, handles), return
end
string ='Enter an energy (in eV) or time (in fs) value that you would like to highlight';
ben=0; if get(handles.bon,'Value'), ben=handles.value.ben; end
xyline = str2num(char(inputdlg(string,'',1,{num2str(abs(ben-handles.value.xyline(1,1)))})));
if ~isempty(xyline)
    answer = questdlg('Is this value in Energy (eV) or Time (fs)','','Energy','Time','Energy');
    if strcmp(answer,'Energy'),  handles.value.xyline = [abs(ben-xyline) 1];
    elseif strcmp(answer,'Time'),handles.value.xyline = [abs(ben-xyline) 2];
    else,return
    end
    set(hObject,'Checked','on')
    guidata(hObject,handles)
    plotscan(hObject, 0, handles)
end

%show Lin/Log division line
function linlogshow_Callback(hObject, eventdata, handles)
if strcmp(get(hObject,'Checked'),'on'),set(hObject,'Checked','off')
else,set(hObject,'Checked','on'),end
plotscan(hObject,0,handles)

function xytick_Callback(hObject, eventdata, handles)
if strcmp(get(hObject,'Checked'),'on'),set(hObject,'Checked','off')
else,set(hObject,'Checked','on'),end
plotscan(hObject,0,handles)


% Plot X bigger than data range
function blankEv_Callback(hObject, eventdata, handles)
%get min and max values
if get(handles.radon,'Value'),   on=1; tx=' radius ';
elseif get(handles.eon,'Value'), on=2; tx=' kinetic energy '; ben=0;
elseif get(handles.bon,'Value'), on=2; tx=' binding energy '; ben=handles.value.ben;
end
if strcmp(get(hObject,'Checked'),'off')
    defans = cellstr(num2str(abs(ben-handles.value.blankEv(on,:)')));%get default values and turn into a cell array of
strings (required format for inputdlg)
    string = {sprintf(['Where would you like your' tx 'axis to start and stop?\n\nstart']),'stop'};
    eV      = str2num(char(inputdlg(string,'',1,defans)));    %get answer and convert from cell array of strings to
numbers
    if isempty(eV), return, end                                %check they have answered, and not pressed cancel or
entered letters instead of numbers
    if ben, eV=ben-eV; end
    handles.value.blankEv(on,:) = eV'; guidata(hObject,handles) %save values
    set(hObject,'Checked','on')
else set(hObject,'Checked','off')
end
plotscan(hObject, 0, handles)
```

```
%No. of fit Graphs
function graph_edit_Callback(hObject, eventdata, handles)
w      = floor(size(handles.value.data,2)/4);
string = ['Input must be between 1 and ' num2str(w)];
txt    = sprintf(['Please Enter number of fit Graphs to display/n/n' string]);
graphs = str2num(char(inputdlg(txt,'',1,{num2str(handles.value.graphs)})));
if isempty(graphs)                                          %if user presses cancel, do nothing
elseif graphs > w || graphs < 1, errordlg(string,'Error'); uiwait   %if user enters dodgy value, throw error
else handles.value.graphs = round(graphs); guidata(hObject,handles) %if good, save value
end
plotscan(hObject,0,handles)                                 %update UI

%% Side Graph Buttons
% 'Reset' Button
function resetsect_Callback(hObject, eventdata, handles)
s.x1=0;s.x2=0;s.y1=0;s.y2=0;
handles.value.scldel = 1;
handles.value.scle   = 1;
handles.value.sidegraphs.s  = s;
set([handles.scldel handles.scle],'String','1')
plotscan(hObject,0,handles)

%'Time Graph Fitting' listbox
function   fitdecay_CreateFcn(hObject,   eventdata,   handles),if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function fitdecay_Callback(hObject, eventdata, handles),
if ~get(handles.fiton,'Value'),set(handles.FitPanel,'Visible','off'),end
plotscan(hObject,0,handles)

% 'Capture' Spectra Graph Button
function snapspectra_Callback(hObject, eventdata, handles)
s=handles.value.sidegraphs.s; x2=handles.value.sidegraphs.x2; y2=handles.value.sidegraphs.y2*handles.value.scle;
s.x2=addd(s.x2,x2,handles.value.stdel); %go to addd function (below) with x values
s.y2=addd(s.y2,y2,handles.value.stpdel);%go to addd function (below) with y values
handles.value.sidegraphs.s=s;           %save data
guidata(hObject,handles)
plotscan(hObject,0,handles)             %plot data

% 'Capture' Time Graph Button
function snapdelay_Callback(hObject, eventdata, handles)
s=handles.value.sidegraphs.s; x1=handles.value.sidegraphs.x1; y1=handles.value.sidegraphs.y1*handles.value.scldel;
s.x1=addd(s.x1,x1,str2double(get(handles.ste, 'String'))); %go to addd function (below) with x values
s.y1=addd(s.y1,y1,str2double(get(handles.stpe,'String'))); %go to addd function (below) with x values
handles.value.sidegraphs.s=s;            %save data
guidata(hObject,handles)
plotscan(hObject,0,handles)              %plot data

% Adds an extra line onto the side plots (called by 'Capture' buttons)
function out=addd(sin,in,value)
if size(in,1)>size(in,2),in=in';end
in=cat(2,value,in);
n=size(sin,2)-size(in,2);
if n==0, out=cat(1,sin,in);
%elseif n >=1, out=cat(1,sin,cat(2,in,zeros(1,n)));
else out=cat(1,cat(2,sin,zeros(size(sin,1),-n)),in);end

% 'Export' Button
function expsect_Callback(hObject, eventdata, handles)
% create variables
handles=guidata(hObject);
[pathname,filename]=fileparts(handles.value.FILE);
x1=handles.value.sidegraphs.x1; y1=handles.value.sidegraphs.y1;
x2=handles.value.sidegraphs.x2; y2=handles.value.sidegraphs.y2;
s=handles.value.sidegraphs.s;
s.x2=addd(s.x2,x2,handles.value.stdel); %go to addd function (below) with x values
s.y2=addd(s.y2,y2,handles.value.stpdel);%go to addd function (below) with y values
s.x1=addd(s.x1,x1,str2double(get(handles.ste, 'String'))); %go to addd function (below) with x values
```

```
s.y1=addd(s.y1,y1,str2double(get(handles.stpe,'String'))); %go to addd function (below) with x values
s.x1=s.x1(2:end,:)';s.y1=s.y1(2:end,:)';s.x2=s.x2(2:end,:)';s.y2=s.y2(2:end,:)'; %get rid of first (nonsense) row, and
turn right way round)
if get(handles.eon, 'Value'), e='K Energy'; u='(eV)'; else e='Time of Flight';u='(ns)'; end
%work out which dataset is longer and set that dateset up to print later
a=size(s.x1); b=size(s.x2);
if a(1)>b(1), x=s.y1; y=s.x1; c=b(1):a(1); d=a(2); q=0; else x=s.x2; y=s.y2; c=a(1):b(1); d=b(2); q=1; end
% open 'save as' box to create file in the same folder as the data came from
[FileName,PathName] = uiputfile('*.txt','Save file as',[pathname '/' filename '_sections']);
if FileName, fid = fopen( fullfile(PathName,FileName) , 'wt');
%Print headings to the new file
    fprintf(fid, ['Sections of File ' filename ', both Delay over Time, and Photoelectron Spectra over ' e '. Created
on' date ' by PlotE \n \n']);
    for z=1:a(2), fprintf(fid,['delays(ps)(' num2str(z) ')  \tcounts(' num2str(z) ')    \t']); end
    for z=1:b(2), fprintf(fid,[e u '(' num2str(z) ')  \tcounts(' num2str(z) ')  \t']); end, fprintf(fid,'\n');
    for z=1:a(2), fprintf(fid,['(' num2str(s.x1(1,z)) u ' to ' num2str(s.y1(1,z)) u ')     \t\t']); end
    for  z=1:b(2),  fprintf(fid,['('  num2str(s.x2(1,z))  '(ps)  to  '  num2str(s.y2(1,z))  '(ps))      \t\t']);  end,
fprintf(fid,'\n');
%print data
    for i=2:min(a(1),b(1)) %print first section whith all columns filled
        for  z=1:a(2),  fprintf(fid,'%-13.2f  ',s.y1(i,z));  fprintf(fid,'\t');  fprintf(fid,'%-13.5f  ',s.x1(i,z));
fprintf(fid,'\t'); end
        for  z=1:b(2),  fprintf(fid,'%-13.2f  ',s.x2(i,z));  fprintf(fid,'\t');  fprintf(fid,'%-13.5f  ',s.y2(i,z));
fprintf(fid,'\t'); end
        fprintf(fid,'\n');
    end
    for i=c %when the shortest columns finish, continue printing the others.
        if q, for z=1:a(2),fprintf(fid,'                \t               \t'); end, end
        for    z=1:d,    fprintf(fid,'%-13.2f    ',x(i,z));    fprintf(fid,'\t');    fprintf(fid,'%-13.5f    ',y(i,z));
fprintf(fid,'\t'); end
        fprintf(fid,'\n');
    end
%finish printing & close file
    fclose(fid)
    set(handles.displaytext,'String',['Sections exported to: ' fullfile(PathName,FileName)])
end
open(fullfile(PathName,FileName));

%% Side Graph Edits
% Energy Start
function    ste_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function ste_Callback(hObject, eventdata, handles)
ste = str2double(get(hObject, 'String'));%get the new stpe value
limits=[get(handles.slidEmax,'Min') get(handles.slidEmax,'Max')];
if isnan(ste) || ste > limits(2) || ste < limits(1)
    set(hObject, 'String', num2str(get(handles.slidEmin,'Value')));%put back to orig. value
    errordlg(['Input must be a number between ' num2str(limits(1)) ' and ' num2str(limits(1)) ],'Error'); uiwait
else set(handles.slidEmin,'Value',ste)
end
plotscan(hObject,0,handles)

%Energy Stop
function    stpe_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function stpe_Callback(hObject, eventdata, handles)
ste = str2double(get(hObject, 'String'));%get the new stpe value
limits=[get(handles.slidEmax,'Min') get(handles.slidEmax,'Max')];
if isnan(ste) || ste > limits(2) || ste < limits(1)
    set(hObject, 'String', num2str(get(handles.slidEmax,'Value')));%put back to orig. value
    errordlg(['Input must be a number between ' num2str(limits(1)) ' and ' num2str(limits(1)) ],'Error'); uiwait
else set(handles.slidEmax,'Value',ste)
end
plotscan(hObject,0,handles)

% Energy Scale
function    scle_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
```

```
function scle_Callback(hObject, eventdata, handles)
value = str2double(get(hObject, 'String'));%get the new value
if isnan(value),set(hObject,'String',handles.value.scle)%put back to orig. value
else handles.value.scle=value; end
guidata(hObject,handles),plotscan(hObject,0,handles)

% Delay Start
function stdel_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function stdel_Callback(hObject, eventdata, handles)
stdel = str2double(get(hObject, 'String'));%get the new stdel value
if isnan(stdel) || stdel > max(handles.value.delmod) || stdel < min(handles.value.delmod)
    set(hObject, 'String', handles.value.stdel);%put back to orig. value
    errordlg(['Input must be a number between ' num2str(min(handles.value.delmod)) ' and '
num2str(max(handles.value.delmod)) ],'Error'); uiwait
else handles.value.stdel = stdel; set(handles.slidDmin,'Value',stdel)
end
guidata(hObject,handles),plotscan(hObject,0,handles)

% Delay Stop
function stpdel_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function stpdel_Callback(hObject, eventdata, handles)
stpdel = str2double(get(hObject, 'String'));%get the new stpdel value
if isnan(stpdel) || stpdel > max(handles.value.delmod) || stpdel < min(handles.value.delmod)
    set(hObject, 'String', handles.value.stpdel);%put back to orig. value
    errordlg(['Input must be a number between ' num2str(min(handles.value.delmod)) ' and '
num2str(max(handles.value.delmod)) ],'Error'); uiwait
else handles.value.stpdel = stpdel; set(handles.slidDmax,'Value',stpdel)
end
guidata(hObject,handles),plotscan(hObject,0,handles)

% Delay Scale
function scldel_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function scldel_Callback(hObject, eventdata, handles)
value = str2double(get(hObject, 'String'));%get the new value
if isnan(value),set(hObject,'String',handles.value.scldel)%put back to orig. value
else handles.value.scldel=value;end
guidata(hObject,handles),plotscan(hObject,0,handles)
%% Sliders
% Side Slider (min)
function slidDmin_CreateFcn(hObject, eventdata, handles),if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function slidDmin_Callback(hObject, eventdata, handles)
handles.value.stdel = get(hObject,'Value');
guidata(hObject,handles), plotscan(hObject,0,handles)
% Side Slider (max)
function slidDmax_CreateFcn(hObject, eventdata, handles),if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function slidDmax_Callback(hObject, eventdata, handles)
handles.value.stpdel = get(hObject,'Value');
guidata(hObject,handles), plotscan(hObject,0,handles)
% Top Slider (min)
function slidEmin_CreateFcn(hObject, eventdata, handles),if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function slidEmin_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)
% Top Slider (max)
function slidEmax_CreateFcn(hObject, eventdata, handles),if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function slidEmax_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)

%% Polar Co-ords Tickbox
function polc_Callback(hObject, eventdata, handles),plotscan(hObject,0,handles)
%% Fitting Stuff
%Main fitting function
function globfit(iterations)
global xofs yofs cct start ncc
```

```
%get variables
handles= guidata(gcbo);
xofs    = handles.value.xofs;    x      = handles.value.y-xofs;
y       = handles.value.z;       yofs   = mean(mean(y(1:3,:)));
f       = handles.value.f;       f      = f(find(f)); %remove zeros
cct     = handles.value.cc;      c      = handles.value.c(:,f)';
lb       =-Inf*ones(size(c));       if get(handles.fitpos,'Value'),lb=0*lb; c(c<0)=0;end %constrain the amplitudes to
positive values if required
lb(:,1)= handles.value.lb(f);    st     = handles.value.consec(f);
ub      = Inf*ones(size(c));     ub(:,1)= handles.value.ub(f);
start   = zeros(length(st),2);   rt     = [0 handles.value.c(1,:)]*1000;
ncc     = 0;                                %always fit using crosscorrelation
for i=1:length(st), b=(st(i)==f+1);          %make rise time (consecutive fitting) matrix
    if any(b)                                %if the user has chosen a channel currently being fitted
        start(i,1)=find(b); start(i,2)=1;    %tell the fit function which channel
        if find(b)==i; start(i,:)=[0 0];end  %if the rise time for a chanel is the same as the decay (i.e. t3 rises
with t3), don't use
    else start(i,1)=rt(st(i)); start(i,2)=0; %if the user has chosen a chanel not currently being fitted, tell the
function the rise time value in fs
    end
end
if any(any(c==1)); c(:,2:end) = ones(length(f),1)*abs(max(y))/(max(max(y))*length(f))/1000; end %if there has been a
change in energy spacing etc, reset all amplitudes to an initial guess based on signal intensity
if max(x)<200, x=x*1000; end                %only work in fs
axes(handles.FitGraph)                      %put the fit stats onto the right axes
%FIT
opt                                  =                  optimset('OutputFcn',@OptPlot,'TolX',1e-20000,'TolFun',1e-
120,'MaxFunEvals',700,'LevenbergMarquardt','on','Display','off');   %set   the   fitting   options   (note:   remove
'Display','off' to see fitting statistics)
for i= 1:iterations, [c,resnorm,residual]=lsqcurvefit(@gdecay,c,x,y,lb,ub,opt); end      %fit using function gdecay
(below)
%save/display values
for i=1:length(f),
    eval(['set(handles.t' num2str(f(i)) ',''String'',' num2str(c(i,1),3) ')'])  %put all the new time values in the
right editboxes
    eval(['set(handles.t' num2str(f(i)) 'fit,''Value'',1)']),   %make sure the right boxes are ticked
end
handles.value.fit     = gdecay(c,x);        %create fit graph using new time constants/amplitudes
handles.value.c(:,f)  = c';                 %save the fitting values for next time
handles.value.residual= residual;           %save the residuals for plotting
guidata(handles.cc,handles)                 %save
plotscan(handles.cc,0,handles)              %go to plot function

%Exponential time decay global fit (called by globfit and tracefit)
function y=gdecay(c,xdata)
global yofs cct Y start ncc
c    = c*1000; sy=Y; xdata=xdata(:);          %c has previosly been devided by 1000 to allow easier fitting (smaller
step sizes)
fits= size(c,1); slices = size(c,2)-1;      %get how many chanels are being used, and how large the dataset is
if slices==1 && c(end)==0, yofs = c(end,1); fits=fits-1; end %if floating y offset
Y    = zeros(length(xdata),slices,fits);    %prepare space for Y (fit for each indevidual channel)
y    = zeros(length(xdata),slices);         %prepare space for y (overall fit data)
rt   = ones (length(xdata),fits);              %prepare space for rt (rise time for each chanel, used in consecutive
fitting)
cc   = ccfit([0 cct 0 1],xdata);                %calculate crosscorelation vector (go to 'Cumulative Gaussian Fit'
function)
if ncc, cc=ones(size(cc)); end              %don't include crosscorelation in beta fit
xdata((xdata)<0) = 0;                       %don't fit before zero with an exponential rise
for f = 1:fits;                             %calculate rise time curves (consecutive dynamics)
    a=0;
    if start(f,2) && start(f,1)~=f, a=c(start(f,1),1);%if rise time is the same as a channel currently being fitted
    elseif start(f,1),a=start(f); end       %if rise time is not the same as a channel currently being fitted
    if a, rt(:,f)=1-exp(-(xdata)./a); end   %if there is a rise time, calculate 1-(e^-tx) rise time curve
end
% yofs=;
for s=1:slices                              %fit each slice (energy bin) of the dataset
    for f = 1:fits; Y(:,s,f) = c(f,s+1).*exp(-(xdata)./c(f,1)).*cc.*rt(:,f); end %fit each chanel (y=A(e^-tx)*cc*rise)
    y(:,s) = sum(Y(:,s,:),3) + yofs;        %add the different chanels together for each slice of data
```

```
end
if slices==1;Y=sy;else Y=Y+yofs; end        %save each chanel (for sidegraph plotting)

% Cumulative Gaussian Fit
function y = ccfit(d,xdata)%calculate the crosscorelation curve (note 2.35482 is the conversion factor between the
gaussian 'varience' and a FWHM)
x = xdata/1000+d(1);
y = 500*d(4)*(1+erf(x/(d(2)*sqrt(2)/2.35482)))+d(3);

%integrated trace fitting function (please note, you can't save things using guidata here
%because it gets called from plotscan which then overwrites the handles structure)
function out=tracefit(x,y,handles)
global cct d c yofs start ncc
if length(y)>5
    %inputs
    txt  = '';    out = [];     xofs = handles.value.xofs;
    yofs = mean(y(1:3));        x    = x(:);            %make sure x is always a column vector
    cct  = handles.value.cc;    opt  = optimset('TolX',1e-20000,'TolFun',1e-1200,'MaxFunEvals',700,'Display','off');
    if max(x)<200, x=x*1000; end                       %only work in fs          ,'LevenbergMarquardt','on'
    %DO FITTING
    %crosscorrelation (gaussian)
    if get(handles.fitdecay,'Value')==2;
        %fit data
        out.c  = fit(x,(y-yofs),fittype('gauss1'));      %calculate fit ('gauss1' uses Y = a1*exp(-((x-b1)/c1)^2) to
fit data)
        out.y  = out.c.a1*exp(-((x-out.c.b1)/out.c.c1).^2)+yofs;%maxe yvalues for plotting fit
        er     = confint(out.c,0.99);                  %get errors (to 99% confidence)
        out.w  = out.c.c1*2.35482/sqrt(2);               %get FWHM (/sqrt(2) gets std dev. *2.354 turns std dev into
FWHM)
        out.cer=(er(2,2)-er(1,2))/2;                   %extract +- error for centre
        %note: fit goodness (including r^2) are stored in out.gof See Help: fit for definitions of what is contained
in this structure
        out.wer=(er(2,3)-er(1,3))*2.35482/sqrt(2);      %extract +- error for FWHM
        txt=['CC='  num2str(out.w*1000,3)  '+-'  num2str(out.wer*1000,2)  ' fs.  Centre='  num2str(out.c.b1*1000,'%1.0f')
'+-' num2str(out.cer/sqrt(2)*1000,'%1.1f') ' fs'];
    %crosscorrelation (rise time)
    elseif get(handles.fitdecay,'Value')==3;
        %d = [-1 0.2 min(y) (max(y)-min(y))/1000];
        yy=sort(y);
        [m i] = max(y); y(i:end) = mean(yy(end-2:end));
        if isempty(d); d = [xofs/1000 cct yofs (max(y)-yofs)/1000]; end
        lb= [-5 0 -Inf 0]; ub= [5 2 Inf Inf];
        d = lsqcurvefit(@ccfit,d,x,y,lb,ub,opt); %fit using
        out.y = ccfit(d,x);
        txt=['CC= ' num2str(d(2)*1000,4) ' fs, xofs= ' num2str(d(1)*-1000,4) ' fs'];
    %Decay Times
    elseif any(get(handles.fitdecay,'Value')==[4 5]);
        set(handles.FitPanel,'Visible','on'), plot(handles.ImageGraph,0),axis(handles.ImageGraph,'off')
        f     = handles.value.f;     f = f(find(f)); %remove zeros
        st    = handles.value.consec(f);
        rt    = [0 handles.value.c(1,:)]*1000;
        ncc   = get(handles.betaon,'Value');%  c     = [];      %don't fit crosscorrelation if beta fit
        if size(c,1)~=length(f) && get(handles.fitdecay,'Value')==5, c = []; end
        if isempty(c), c = handles.value.c(1:2,f)'; c(:,2) = abs(1/(1000*length(f))); end
        if size(c,1)==length(f) && get(handles.fitdecay,'Value')==4, c(end+1,1) = yofs/1000; end
        lb    =-Inf*ones(size(c));   lb(1:length(f),1)= handles.value.lb(f)';
        ub    = Inf*ones(size(c));   ub(1:length(f),1)= handles.value.ub(f)';
        for i=1:length(st), b=(st(i)==f+1);         %make rise time (consecutive fitting) matrix
            if any(b)                               %if the user has chosen a channel currently being fitted
                start(i,1)=find(b); start(i,2)=1;   %tell the fit function which channel
                if find(b)==i; start(i,:)=[0 0];end %if the rise time for a chanel is the same as the decay (i.e. t3
rises with t3), don't use
            else start(i,1)=rt(st(i)); start(i,2)=0; %if the user has chosen a chanel not currently being fitted, tell
the function the rise time value in fs
            end
        end, start(end+1,:)=[cct 0];
        for i=1:1, [c err]=lsqcurvefit(@gdecay,c,x,y,lb,ub,opt); end %fit using function tdecay
        out.y = gdecay(c,x); out.in=y;
```

```
        for i=1:length(f), txt=[txt ' t' num2str(f(i)) '=' num2str(c(i,1),3) ' ps,'];end
        txt=[txt ' err=' num2str(err,3)];
    end
else txt='Not enough data points';out=[];
end
set(handles.fittxt,'String',txt)

%set up Optimisation Plot
function setupoptplot(handles)%setup the figure and make it's axes look right
axes(handles.FitGraph), h = plot(0,0); xlim([0 100])
set(h,'Tag','optplot');  set(gca,'YTickLabel',[])

%Optimisation Plot Function (called on every iteration of lsqcurvefit in globfit)
function s = OptPlot(x,optimValues,state)
%handles=guidata(gcf);axes(handles.FitGraph)
if optimValues.iteration == 0 %setup stuff before we start
    plotresnorm = findobj(get(gca,'Children'),'Tag','optplot');
    Y = get(plotresnorm,'Ydata');
    if Y==0; set(plotresnorm,'Ydata',optimValues.resnorm)%get rid of 0 point at beginning of new graph
    elseif length(Y)>90 %if too long
        newY = get(plotresnorm,'Ydata'); set(plotresnorm,'Ydata',newY(end-71:end)); %only use the most recent values
    end
else
    plotresnorm = findobj(get(gca,'Children'),'Tag','optplot'); %get all the points already plotted
    newY = [get(plotresnorm,'Ydata') optimValues.resnorm];      %add on the most recent value
    set(plotresnorm,'Xdata',1:length(newY),'Ydata',newY);       %add it to the plot
    set(get(gca,'XLabel'),'String',num2str(optimValues.resnorm,4));%display it below the plot
end
s = false;

%% FITTING Sectionbox (Top Left Section with loads of Editboxes)
%FUNCTIONS CALLED BY UI CONTROLS
function fitscan(handles) %fitting function
global c; c=[];
if get(handles.fiton, 'Value'), globfit(1), %if viewing global fit
else plotscan(handles.cc,0,handles), end    %if fit decays only

%'Fit' Tick Box Function
function fitselect(hObject,num,handles)%fitting tickbox
handles.value.f(num) = get(hObject,'Value')*num;    %get the current tickbox value
handles.value.c(2:end,:) = 0;       %reset amplitudes
if ~any(handles.value.f), set(hObject,'Value',1),errordlg('You must have at least one box ticked') %check that there
is at least one box ticked
else guidata(hObject,handles),fitscan(handles)  %if there is, save values and fit.
    for i=1:4, eval(['set(handles.t' num2str(i) 'fit,''Value'',' num2str(handles.value.f(i)/i) ')']),end
end %^check what boxes are ticked and make sure they are showing as ticked

%'Fix' Tick Box Function
function varyselect(hObject,num,handles)%fix tickbox
tbhandle=eval(['handles.t' num2str(num) 'vary']);
val=handles.value.c(1,num);
if get(tbhandle,'Value'),lb=val; ub=val+0.00001;
elseif tbhandle==hObject, lb=0; ub=Inf;
else lb=handles.value.lb(num);ub=handles.value.ub(num);end
handles.value.lb(num)=lb; handles.value.ub(num)=ub;
handles.value.lb
guidata(hObject,handles), fitscan(handles)  %save values and fit.
for i=1:4, eval(['set(handles.t' num2str(i) 'vary,''Value'',' num2str(handles.value.lb(i)~=0) ')']),end
%^check what boxes are ticked and make sure they are showing as ticked

%Start from' listbox function
function consecselect(hObject,num,handles)%fitting tickbox
handles.value.consec(num) = get(hObject,'Value');    %get the current listbox value
guidata(hObject,handles),fitscan(handles)  %if there is, save values and fit.
for i=1:4, eval(['set(handles.t' num2str(i) 'start,''Value'',' num2str(handles.value.consec(i)) ')']),end
%^check what boxes are ticked and make sure they are showing as ticked

%Edit Box function
```

```
function value=fitedt(hObject,hValue)
global c; c=[];
v=str2double(get(hObject,'String'));    %get the value
if isnan(v),set(hObject,'String',hValue),errordlg('value must be a number'),value=[];%check it's good
else value=v; end                       %send it back to calling function to be saved

%UI CONTROLS
%Make Amps Positive Tickbox
function fitpos_Callback(hObject, eventdata, handles),fitscan(handles)
% Amplitudes tickbox
function fitamps_Callback(hObject,eventdata,handles),plotscan(hObject,0,handles)

% function fitgraphs_Callback(hObject, eventdata, handles), plotscan(hObject,0,handles)

%'Fix' Tick Boxes
function t1vary_Callback(hObject, eventdata, handles),varyselect(hObject,1,handles)
function t2vary_Callback(hObject, eventdata, handles),varyselect(hObject,2,handles)
function t3vary_Callback(hObject, eventdata, handles),varyselect(hObject,3,handles)
function t4vary_Callback(hObject, eventdata, handles),varyselect(hObject,4,handles)

%'Fit' Tick Boxes
function t1fit_Callback(hObject, eventdata, handles),fitselect(hObject,1,handles)
function t2fit_Callback(hObject, eventdata, handles),fitselect(hObject,2,handles)
function t3fit_Callback(hObject, eventdata, handles),fitselect(hObject,3,handles)
function t4fit_Callback(hObject, eventdata, handles),fitselect(hObject,4,handles)

%'Start From' listboxes
function t1start_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t2start_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t3start_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t4start_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t2start_Callback(hObject, eventdata, handles),consecselect(hObject,2,handles)
function t1start_Callback(hObject, eventdata, handles),consecselect(hObject,1,handles)
function t3start_Callback(hObject, eventdata, handles),consecselect(hObject,3,handles)
function t4start_Callback(hObject, eventdata, handles),consecselect(hObject,4,handles)

%cross correlation
function cc_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function cc_Callback(hObject, eventdata, handles)
global d,d=[]; value=fitedt(hObject,handles.value.cc); %go to fitedt function (above) to get value
if ~isempty(value), handles.value.cc = value; guidata(hObject,handles),fitscan(handles),end

%'Time (ps)' Editboxes:
function t1_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t2_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t3_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t4_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function t1_Callback(hObject, eventdata, handles)
value=fitedt(hObject,handles.value.c(1,1)); %go to fitedt function (above) to get value
if ~isempty(value), handles.value.c(1,1) = value; guidata(hObject,handles),varyselect(hObject,1,handles),end
function t2_Callback(hObject, eventdata, handles)
value=fitedt(hObject,handles.value.c(1,2)); %go to fitedt function (above) to get value
if ~isempty(value), handles.value.c(1,2) = value; guidata(hObject,handles),varyselect(hObject,2,handles),end
function t3_Callback(hObject, eventdata, handles)
value=fitedt(hObject,handles.value.c(1,3)); %go to fitedt function (above) to get value
if ~isempty(value), handles.value.c(1,3) = value; guidata(hObject,handles),varyselect(hObject,3,handles),end
function t4_Callback(hObject, eventdata, handles)
value=fitedt(hObject,handles.value.c(1,4)); %go to fitedt function (above) to get value
if ~isempty(value), handles.value.c(1,4) = value; guidata(hObject,handles),varyselect(hObject,4,handles),end
```

```
%% (tof2ke) Kinetic Energy Binning
function [eNe e calib_file new] = tof2ke(handles,data)

%retreave useful variables
ftype      = handles.value.ftype;
emin       = str2num(get(handles.emin,'String'));       %starting point (ke axis)
% emax       = handles.value.emax;       %ending point (ke axis)
emax       = str2num(get(handles.emax,'String'));
e_points   = handles.value.e_points;   %ke resolution
new        = 0;
Re         = handles.value.Re;
%etof_res_ns = handles.value.etof_res_ns;
%delays     = handles.value.delays*1000; %x axis

%If data=1 viewing *dat file and no tof2ke needed
f=1;
if strcmp(ftype,'edat')
    eNe=handles.value.eNe;  e=handles.value.e;  calib_file='No calibration file required - viewing an energy *.dat
file';
    if e(end)>emin || e(1)<emax,e=fliplr(e);end
    while e(end)<=emin,e=e(1:end-1);eNe=eNe(:,1:end-1);end
    while e(1)>=emax,e=e(f:end);eNe=eNe(:,f:end);f=f+1;end
    return
elseif any(strcmp(handles.value.calib_file,{'','No calibration file required - viewing *.dat file'}))
    [PathName]=fileparts(handles.value.FILE);
    [FileName,PathName] = uigetfile('*.cal','Choose your calibration file',[PathName '\Calibration.txt']);
    calib_file=fullfile(PathName,FileName); new=1;
else
    calib_file=handles.value.calib_file;
end
%do the conversion
e = wrev(emin:e_points:emax);        %create energy vector
[eNe e]=econvert(data,calib_file,e,Re);
%update UI and make sure data is good
if  new==1;set(handles.emax,'String',max(e)),set(handles.emin,'String',min(e)),end  %if  it's  a  new  calibration  file,
reset the min and max energy editboxes
eNe(isnan(eNe))=0;%get rid of NAN errors


%convert radius into energy
function [odata e]=econvert(idata,Cfile,e,R)
%idata   = data to be converted
%Cfile   = calibration file
%binwidth= energy resolution
%R       = original radius values for data
%odata   = converted dataset
%e       = new energy values for data
N    = size(idata,1);
P    = size(idata,3);
fid  = fopen(Cfile,'r');
buf  = textscan(fid,'%s',3,'delimiter','\n','commentStyle','%');
calib= buf{1};
A    = sscanf(calib{1},'A=%f');
E0   = sscanf(calib{2},'E0=%f');
t0   = sscanf(calib{3},'t0=%f');
fclose(fid);
%create the arrays and values
emax = ((R(end)-t0)^2)/A+E0-0.1;
emin = min(e);
width= abs(e(1)-e(2));
e    = wrev(emin:width:emax);
% e                = wrev(emin:binwidth:emax);        %create energy vector
eT_rebinned       = real(sqrt(A.*(e-E0))+t0);   %calculate the radius at each energy
% eT_rebinned(end)  = 1;
eT_rebinned_index = eT_rebinned/(R(6)-R(5));    %calculate where in the radius vector each energy point lies
odata             = zeros(N,length(e));         %prepare the space for the new energy resolved data set
```

```
%rebin it
for k=1:P                                   %if there is more than one dataset
    for i = 1:N                             %for each time or angle step
        jt=R(end); jtt=jt;                  %start point (max radius)
        for j=1:length(e)                   %for each energy step
            summ = 0;                       %reset data point value
            jbb  = eT_rebinned_index(j);    %get min radius value
            jb   = floor(eT_rebinned_index(j)); %get rounded min radius value
            summ = summ + sum(idata(i,jb+2:jt,k),2);
            summ = summ + (jb+1-jbb)*idata(i,jb,k) + (jtt-jt)*idata(i,jt,k);
            if jb==jt; summ= (summ-1*idata(i,jb,k)); end
            jtt  = jbb; jt = jb;            %make the next step's max value equal to this steps min value
            odata(i,j,k) = summ;
        end
    end
end
odata = flipdim(odata,2);
e     = wrev(e);

% use these lines to convert between specific energy or radius values and
% write resultant value to screen
% radius = 135;
% energy = ((radius-t0)^2)/A+E0
% energy = 0.244;
% radius = sqrt(A.*(energy-E0))+t0
%% Change Time Axis
function deledit(varargin)                  %apply a multiplication factor to time delays in a saved file
[Norm Good] = getpath;                      %get the file path name
if isempty(varargin)                        %get the file to be modified if not specified
     [name path]=uigetfile({'*.dat','Quick Save Data (*.dat)';'.mat','Full Save Data (*processed.mat)'},'Pick a file
to Modify',[Good '*processed.dat']);
     file = [path name];
else file = char(varargin);
end
%get the multiplication factor
mod = str2num(char(inputdlg('What would you like to times your time axis by?','',1,{'-1'})));
%if its a .mat file:
if ~isempty(mod) && strcmp('.mat',file(end-3:end))
    load(file)                              %load the file
    delays=delays*mod;                      %#ok<NODEF,NASGU> modify delays
    [name path]=uiputfile({'*.mat','Full Save Format (*.mat)'},'Pick new name for Modified file',[file(1:end-13) '2'
file(end-13:end)]);
    if name~=0;
        save([path name],'delays','intdata','beta','rad','imdata','Isize','scantrace','scanrange')
    end
% if it's a .dat file:
elseif ~isempty(mod) && strcmp('.dat',file(end-3:end))
    fid = fopen(file,'r');                  %open the file
    f   = fscanf(fid,'%c');                 %get the text
    fclose(fid);                            %close the file
    %get new file to save modified delays to
    [name path] = uiputfile({'*.dat','Quick Save Format (*.dat)'},'Pick new name for Modified file',[file(1:end-13)
'2' file(end-13:end)]);
    if name==0;return,end                   %if user presses cancel
    p = findstr(f,'schritte'); n = str2num(f(p+10:p+14));%get no. of delay positions
%     p = findstr(f,'X-scale');
%     p(1) = findstr(f,'Wellenlaengen')+18; p(2) = findstr(f,'XMode')-2; %find energy spacing
    p = findstr(f,'#')+1;                   %find main data
    l = findstr(f(p(3):end),sprintf('\n'))+p(3)-1;%get all the linebreaks (so you can just get the correct number of
lines)
    a = str2num(f(l(1):l(n+1)));            %extract main dataset
    p = findstr(f,'*')+1;                   %find background data (if it exists)
    for i=1:length(p)
        l = findstr(f(p(i):end),sprintf('\n'))+p(i)-1;   %get all the linebreaks (so you can just get the correct
number of lines)
        b(:,:,i) = str2num(f(l(3):l(n+4)));     %extract background dataset
    end
    %get extracted data
```

```
    if isempty(b)                               %if no background data
        bRad= []; cRad = []; p = findstr(f,'#')+1;
    else                                        %get background data
        bRad= b(:,2:end,1); cRad = b(:,2:end,2);
    end
    rRad    = a(:,2:end);                        %get main data
    delays  = a(:,1)*mod;                        %MAKE MODIFICATION
    %save to new file
    fid     = fopen([path name],'wt');           %open new file for writing to
    fprintf(fid,f(1:p(1)))                      %write the headers to the file unmodified
    delaysN = length(delays);                    %get the number of lines to write for each dataset
    if  any(any(any(bRad)))
        fprintf(fid, '*\n\n Pump Alone Data = \n\n');%write Pump Alone Data
        for i=1:delaysN
            %Every line starts with delay in femtosec which is followed by corresponding PES
            fprintf(fid, '%6.4f ', delays(i)); fprintf(fid, '%6.4f ', bRad(i,:)); fprintf(fid, '\n');
        end;
    end
    if  any(any(any(cRad)))
        fprintf(fid, '*\n\n Probe Alone Data = \n\n');
        for i=1:delaysN
            %Every line starts with delay in femtosec which is followed by corresponding PES
            fprintf(fid, '%6.4f ', delays(i)); fprintf(fid, '%6.4f ', cRad(i,:)); fprintf(fid, '\n');
        end;
    end
    fprintf(fid, 'Data = \n#\n');
    for i=1:delaysN
        %Every line starts with delay in femtosec which is followed by corresponding PES
        fprintf(fid, '%6.4f ', delays(i)); fprintf(fid, '%6.4f ', rRad(i,:)); fprintf(fid, '\n');
    end;
    fclose(fid); plotE_1_0([path name]);                        %open new file in PlotE
end
%% herrorbars (horazontal error bars)
function hh = herrorbar(x,y,l,u,symbol)
if min(size(x))==1, npt = length(x); x = x(:); y = y(:);
    if nargin > 2,
        if ~ischar(l),l = l(:);end
        if nargin > 3, if ~ischar(u),u = u(:); end, end
    end
else,[npt,n] = size(x); %#ok<NASGU>
end
if nargin == 3
    if ~ischar(l),u = l;symbol = '-';
    else,symbol = l; l = y; u = y; y = x; [m,n] = size(y); x(:) = (1:npt)'*ones(1,n);end
end
if nargin == 4
    if ischar(u),symbol = u;u = l;
    else,symbol = '-';end
end
if nargin == 2
    l = y; u = y; y = x;[m,n] = size(y);
    x(:) = (1:npt)'*ones(1,n); symbol = '-';
end
u = abs(u);l = abs(l);
if ischar(x) | ischar(y) | ischar(u) | ischar(l),error('Arguments must be numeric.'),end %#ok<OR2>
if ~isequal(size(x),size(y)) | ~isequal(size(x),size(l)) | ~isequal(size(x),size(u)), %#ok<OR2>
    error('The sizes of X, Y, L and U must be the same.');
end
tee= (max(y(:))-min(y(:)))/100; % make tee .02 x-distance for error bars
xl = x - l;  ytop = y + tee;
xr = x + u;  ybot = y - tee;
n  = size(y,2);
% Plot graph and bars
hold_state = ishold;
cax  = newplot; %#ok<NASGU>
% next = lower(get(cax,'NextPlot'));
% build up nan-separated vector for bars
xb = zeros(npt*9,n);xb(1:9:end,:) = xl;xb(2:9:end,:) = xl;
```

```
xb(3:9:end,:) = NaN;xb(4:9:end,:) = xl;xb(5:9:end,:) = xr;
xb(6:9:end,:) = NaN;xb(7:9:end,:) = xr;xb(8:9:end,:) = xr;
xb(9:9:end,:) = NaN;
yb = zeros(npt*9,n);yb(1:9:end,:) = ytop;yb(2:9:end,:) = ybot;
yb(3:9:end,:) = NaN;yb(4:9:end,:) = y;yb(5:9:end,:) = y;
yb(6:9:end,:) = NaN;yb(7:9:end,:) = ytop;yb(8:9:end,:) = ybot;
yb(9:9:end,:) = NaN;
[ls,col,mark,msg] = colstyle(symbol); if ~isempty(msg), error(msg); end
symbol = [ls mark col]; % Use marker only on data part
esymbol = ['-' col]; % Make sure bars are solid
h = plot(xb,yb,esymbol); hold on
h = [h;plot(x,y,symbol)];
if ~hold_state, hold off; end
if nargout>0, hh = h; end

%% Help Menu

function HelpMenu_Callback(hObject, eventdata, handles)
function Abhelp_Callback(hObject, eventdata, handles), fhelp('About')
function PEHelp_Callback(hObject, eventdata, handles), fhelp('PlotE Help')
function PGHelp_Callback(hObject, eventdata, handles), fhelp('Programming Help')
function Xhelp_Callback(hObject, eventdata, handles),  fhelp('X Axis Help')
function Yhelp_Callback(hObject, eventdata, handles),  fhelp('Y Axis Help')
function Zhelp_Callback(hObject, eventdata, handles),  fhelp('Z Axis Help')
function Apphelp_Callback(hObject, eventdata, handles),fhelp('Appearence Help')
function SGhelp_Callback(hObject, eventdata, handles), fhelp('Side Graph Help')
function PrismCompressor_Callback(hObject, eventdata, handles),prismcomplength
function BKWTunnelRate_Callback(hObject, eventdata, handles),BKWTunneling

function fhelp(subject)
file='plotE_1_0';
if strcmpi(subject,'X Axis Help')
    helptxt=['The X Axis Section allows you to modify the units and extent'...
    ' of the X Axis of the main graph. If you choose Radius then the graph'...
    ' will be displayed in units of distence (in pixels) from the centre of'...
    ' the image. If you choose Kinetic Energy, the graph will be displayed'...
    ' in units of electron kinetic energy. You can change the minimum and'...
    ' maximum vaules to zoom in or out. The Energy Division box lets you'...
    ' change the bin width of the calculation. Electron Kinetic Energy is'...
    ' proportional to the radius squared. When you first click on Kinetic'...
    ' Energy you will be asked to choose a calibration file. If you want to'...
    ' make a new calibration file, load up data of a molecule where you know'...
    ' what energy the peaks should lie at, then choose "Create Calibration'...
    ' File" from the Data Menu. '];
elseif strcmpi(subject,'Y Axis Help')
    helptxt=['The Y Axis Section allows you to modify the units and extent'...
    ' of the Y Axis of the main graph. You can change the minimum and'...
    ' maximum vaules to zoom in or out. You can reverse the axis, in case the'...
    ' time delay has been recorded backwards. You can display the graph'...
    ' on a linear/logarithmic axis, where the first section is linear,'...
    ' but the larger time delays are plotted on a logarithmic scale. The'...
    ' editbox beside this option allows you to choose where you switch from'...
    ' a linear scale to a logarithmic scale. Please note, many datasets are'...
    ' recorded using linear-logaritmic spacing for the time delays. To see'...
    ' where the data changes scale, the "Mesh" option in Appearence can be'...
    ' helpful'];
elseif strcmpi(subject,'Z Axis Help')
    helptxt=['The Z Axis Section allows you to modify the units and extent'...
    ' of the Z Axis of the main graph. The Z axis controls the coloring in'...
    ' 2D mode, and the height in 3D mode.\n\n'...
    ' The Intensity option displays the Electron Intensity at'...
    ' that radius/energy vs time. You can choose to subtract'...
    ' r^2 noise. Since the circular integration process adds over a larger'...
    ' area at larger radeii, average background noise can increase at larger'...
    ' radeii. This option fits a r^2 line to the t<0 data and subtracts it'...
    ' from all the data.\n\n'...
    ' The Anisotropy option displays the calculated Beta parameter (a measure'...
    ' of anisotropy ranging from -1 to 2) for each radius/energy vs time. The'...
```

```
    ' Anisotropy parameter is calculated by fitting a sin^2 graph to the angular'...
    ' data. The beta*inten tickbox lets you multiply the beta parameter with the '...
    ' Intensity parameter required to fit the data, this helps to isolate the regions'...
    ' of interest.\n\n'...
    ' You can see the anisotropy fit by selecting "display fit". This shows a'...
    ' series of graphs (the number of graphs can be changed in the editbox). All'...
    ' the graphs refer to one image, selected using the side slider and displayed'...
    ' in the top left image. If 10 graphs are shown, the polar image data is sliced'...
    ' into 10 slice of equal (radius) width. In each slice, the program chooses'...
    ' the most interesting radius value, and shows the angular distribution'...
    ' and the fit at that value. Changing the top sliders means that you can'...
    ' select the radius values that you are most interested in. If you are'...
    ' unhappy with the fit you can improve it using the improve fit button. If'...
    ' you use this, remember to save the fit using the orange "Save fit" button'...
    ' that appears once the fit is complete. The Export Fits button lets you export'...
    ' all the data shown in the graphs into a text file that is formatted to be easily'...
    ' copied into a spreadsheet.'];
elseif strcmpi(subject,'Appearence Help')
    helptxt='hello! 3';
elseif strcmpi(subject,'Side Graph Help')
    helptxt='hello! 4';
elseif strcmpi(subject,'About')
    helptxt=['This software was written by Ruth Livingstone over the course of'...
    ' her PhD. It is designed to open data files created by Process_Data.m .'...
    ' These files should be in the form of a processed.mat file containing'...
    ' all the variables required. It was written using matlab 7.4.0(R2007a)'];
elseif strcmpi(subject,'Programming Help')
    helptxt=['Programming help is located at the beginning of the file'...
    ' "PlotEv8.m" All the user interface (UI) and variable information'...
    ' is stored there. To edit the user interface, type "guide"'...
    ' into the command line, and open "' file '.fig" when prompted.'];
elseif strcmpi(subject,'PlotE Help')
    a=help(file);%get the first few commented lines from Camera_Stage.m
    b=isspace(a);a(b)=' ';%get rid of artificial newlines so text wraps nicely
    c=strfind(a,a(1:5));%get rid of first few lines of nonsense(if they exist)
    c(end+1:end+3)=1;%if no lines on nonsense, start from beginning
    helptxt=sprintf(a(c(3):end));
else helptxt=['Sorry. No help files written yet for ' subject];
end

helpdlg(sprintf([subject '\n\n' helptxt]),subject);
```

Appendix.B. Acquire Data

```
function varargout = Aquire_Data_1_0(varargin)
%Controls hardware and collects Photoelectron Images
%This program connects to a camera, two shutters
%and a stage in order to collect data while changing the time delay
%between two femtosecond (fs) pulses. All the stage settings are in fs
%rather than mm since the purpose of moving the stage is to change the
%time delay between the two pulses. The two shutters cut off each of the
%two pulses in order to allow background images to be obtained. The
%images are collected, and some very simple processing is done on them
%to enable a crude spectra vs time delay graph to be shown during the
%data collection. Each collected image is saved in a mat file. Be aware that
%these files take up a lot of memory and you might want to get rid of
%unwanted files every now and then to clear up computer space. If you want
%help on a specific topic then please choose the topic from the help menu.\n\n
%The Run button will set a Run into motion, using all the settings
%currently displayed. Please avoid changing the settings or moving the
%stage while a run is in process. Abort Run will stop the scan straight
%away. All the images before that point will still be saved. Pause Run
%stops the run at the end of the next scan. Pressing Pause Run again will
%re-start the Run at the beginning of the next run.
%The Live Camera View button gives you a live camera image, and a graph of the
%total signal vs time.
s='Please note, the above text is displayed in "General Help" in the Help Menu';
%todo:
% fix preview popup figure
% display a warning if delays goes out of range for the stage

%% Programming help:
%If you need to change the program for any reason, this section is here to
%try and help you.

%VARIABLES
%The main variables are all saved as handles.value.variable and a full
%list of these are shown below. In addition to these, A few variables are
%set as global variables and are used mainly in the Run function and the
%functions called by Run.

%CHANGING USER INTERFACE:
%To open the user interface editor, type "guide" into the command window,
%choose "Open Existing GUI" and choose Aquire_Data_1_0.fig. The figure should
%open in editing mode. If you double click any of the controls you get a
%list of properties. The important general properties are:
%String: This is what is written on the control (i.e. what the button says)
%Tag: This is what you use when programming to identify the control
%TooltipString: This is what you see if you hover your mouse over the control
%Visible: This allows you to hide or show the control.
%Enable: this allows you to grey out the control, so it is still visible
%but cannot be used.
%There are other properties such as value, listbox top etc that are
%important only for specific controls. Use the MATLAB help files if you are
%stuck on these.

%All the properties (apart from Tag) can be accessed and changed
%programmatically using get(handles.Tag, 'property') and
%set(handles.Tag,'property',new_value). All the existing Tags are listed
%above, along with a brief description of what they are.

%To make a new control(button/graph/editbox etc) choose the type of
%control you want from the bar at the left and place it with the mouse.
%Alter the properties by double clicking the new control. Once you save it,
%MATLAB will generate some code at the bottom of this program. Ignore
%the 'CreateFcn' if generated (don't delete it though!). The 'Callback'
%function gives you the opportunity to write script that executes only if
%the user uses the control. Have a look through this program to see some
%examples of Callback routines for the type of control you want.
```

```
%note: guidata(hObject,handles) saves any handles.value variables

%CHANGING HARDWARE:
%Stage:
%If you change the Stage, you will need to change the part number in the
%"Set up Stage" function (err=setupstage(handles,location)). The part no.
%looks like 'M-403.62S' and should be written on the stage.
%IMPORTANT: our stage is wonky and moves half as far as you ask it to. If
%you change stage, please replace all the '0.0002998' conversion factors
%(from fs to mm) to '0.00015' to reflect the double pass path length change

%Shutters:
%If you change the shutters you will need to change the settings of the
%ActiveX Controls. To do this open the figure in Guide, set "GUI Options:
%Resize Behavior" to "Non-Resizable". Make the figure larger by dragging
%the corner to the right. You should see 2 small white squares. If you
%double click these you get the settings. The HWSerialNumber is what you
%want to change. It should look like 85819025 and on the shutter control
%box it should be written as S/N85819025. Change the number to match the
%control box, leaving out the S/N. Remember to do the same thing for the
%Test Hardware GUI as well.

%Camera:
%If you change the camera with the same type of camera there shouldn't be
%any problem, it should work fine. If you change it to another type of
%camera then be careful as it might only work some of the time depending on
%what "Camera Settings" you put in. Also, the frame rate choice might not
%work properly (i.e. the camera might work, but not be operating on the
%framerate you think it is). You might need to change the Contrast editbox
%to accept/reject different values (camcontrast_Callback), and the
%framerate listbox to give different choices (use guide, see 'changing user
%interface' above)).
%if the camera doesn't work at all, try the image acquisition help files,
%and try to get it working from the command line or with a small m file.
%Once you have it working, a search for 'vid = videoinput('winvideo',1);'
%should get you to every occasion I setup the camera.

%end of programming help. Good luck!

%% Setup for user interface (gui). **DO NOT EDIT!**
% Last Modified by GUIDE v2.5 21-Mar-2012 18:09:17
        gui_Singleton = 1;
        gui_State = struct('gui_Name',       mfilename, ...
                           'gui_Singleton',  gui_Singleton, ...
                           'gui_OpeningFcn', @Aquire_Data_1_0_OpeningFcn, ...
                           'gui_OutputFcn',  @Aquire_Data_1_0_OutputFcn, ...
                           'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
        if nargin && ischar(varargin{1})
            gui_State.gui_Callback = str2func(varargin{1});
        end

        if nargout
            [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
        else
            gui_mainfcn(gui_State, varargin{:});
        end
% End of DO NOT EDIT!
%% Commands to be run before the user interface opens

function Aquire_Data_1_0_OpeningFcn(hObject, eventdata, handles, varargin)
global trace
trace=1;

%Setup handles and related structure:
    handles.output = hObject;
    guidata(hObject, handles);
```

```
%Check what the date is(for writing filenames.):
    Date=0;                 %in case 'name.mat' has been deleted/written over
    load name
    if strcmp(datestr(date,'yymmdd'),Date)==0
        Count=1;            %if it's a new day, restart the count at 1
        Date=datestr(date,'yymmdd');
    end
    save('name','Date','Count')

%Open the file 'Default_Settings' and write the settings to the UI:
    PathName = 'Settings/'; FileName = 'Default_Settings';
    Load_Settings(hObject,handles,FileName,PathName)% go to Load_Settings Fn

% Outputs from this function are returned to the command line
function varargout = Aquire_Data_1_0_OutputFcn(hObject, eventdata, handles)
global stage datap maskc cropd cropt c maskt cl
stage='';datap=0;tic;maskc=0;cropd=0;cropt=0;c=0;maskt=0;

varargout{1} = handles.output;

%take initial pic & graph:
vid = videoinput('winvideo',1);  set(vid,'FramesPerTrigger',Inf);
src=getselectedsource(vid);      camcontrast=set(src,'ContrastMode');
frame=set(src,'FrameRate');      src.ContrastMode = camcontrast{2};
%get the changeable values (Aquire time, Frame rate, Contrast, centre)
load dataf
src.Contrast = handles.value.camcontrast;
src.FrameRate = frame{handles.value.framerate};
handles.value.piccentre=eval(['[' get(handles.piccentretxt,'String') ']']);
switch handles.value.framerate,case 1,framerate=60;case 2,framerate=30;case 3,framerate=15;case 4,framerate=7.5;case
5,framerate=3.75;end
% m=ceil(str2double(get(handles.imagetime,'String'))*framerate);
m=round(framerate/3);  %1/3 of a second

%get the picture
tdata = 0; cl=0;
try
    start(vid)
    rdata = getdata(vid,5);         %throw away first few frames
    rdata = getdata(vid,m);         %get all the frames
    tdata  = mean(rdata,4);         %add them together
    stop(vid)
    flushdata(vid)
    data=tdata; save('dataf','data','wd');stop(vid)
    picdisp1(data,handles,[num2str(m) ' shots per picture.'])

catch %if the camera isn't working or plugged in
    err = lasterror;
    set(handles.textdisplay,'String',['Picture Error: ' err.message])
    Error_on_line=err.stack.line
    disp(err.message)
end
handles.value.shutters=0;
handles.value.c=0;

guidata(hObject,handles)
%% '
%% MAIN BUTTONS & FUNCTIONS
%% Run Buttons
function ExitScan_Callback(hObject, eventdata, handles)
global ExitScan, ExitScan=1;

function PauseScan_Callback(hObject, eventdata, handles)
global ExitScan
if get(hObject,'Value')
    answer=questdlg('Scan has been paused. Would you like to keep running, and pause or at the end of the next
scan?','Pause Run','Finished','End of Scan','End of Scan');
    if strcmp(answer,'Finished'),set(hObject,'Value',0)
```

```
    elseif strcmp(answer,'End of Scan')
        set(handles.gaustxt,'string','Run will pause once scan has finished...')
        set(hObject,'Enable','off'),ExitScan=2;
    end
else
    set(handles.gaustxt,'string','')
    ExitScan=3; runn(hObject, handles);
end

function Run_Callback(hObject, eventdata, handles)
fclose('all'),runn(hObject, handles);

%% 'Run' Function Setup
function runn(hObject,handles)
%Exitscan = 0 (default, everything running well)
%Exitscan = 1 (Abort scan pressed, window attempted to close, or hardware malfunction. Stop Run immediately)
%Exitscan = 2 (Pause button pressed, exit Run at end of scan, but don't close down hardware)
%Exitscan = 3 (Pause button pressed to restart Run. Set Run going at start of next scan. Don't try to initialise
hardware)
%setup various parameters and variables
global delays stage Running barh stageaxis vid path scan ExitScan name iset err imthresh csat tdata sccomp cl centon
persistent bg is ha
err=[];
try
load dataf, load name,
D=datestr(date,'yymmdd'); iset=0; sc=1; filename=''; pic=0; sherr=0;
if strcmp(D,Date)==0                                %Check that the date is still correct
    Count=1; Date=D; set(handles.runcounter,'String',Count); %if it's a new day, restart the count at 1
end

%Setup Stage part 1:
ferr=setupstage(handles,0);                         %go to setupstage function

%calculate variables
dsign     = 1-2*get(handles.scanbackwards,'Value'); %1 if no backwards scan. -1 if yes backwards scan
centon    = get(handles.CentroidIm,'Value');
delays    = handles.value.delays; imagenum=length(delays);
framerate = 120/(2^handles.value.framerate);        %turns 1,2,3... into 60,30,15...
shotspp   = round(handles.value.imagetime*framerate); if shotspp==0, shotspp=1; end
shotsb    = round(handles.value.btime*framerate);     if shotsb==0, shotsb=1;   end
shotsc    = round(handles.value.ctime*framerate);     if shotsc==0, shotsc=1;   end

if ExitScan==3||ExitScan==1,sc=scan;                %if Pause is trying to restart scan, don't restart the hardware.
else
    Running=1;
    col=get(handles.figure1,'Color');
    barh=waitbar(0.1,'Please wait, setting up hardware...','CreateCancelBtn',...
        'global                                     ExitScan,                                           ExitScan=1;
delete(gcbf)','CloseRequestFcn','delete(gcbf)','Color',col,'Visible','off');%create the wait bar in bottom corner
    set(barh,'Position',[0 0 270 80],'Visible','on')
%turn off & manage user interface controls
    ha = findobj(handles.figure1,'Enable','on');    %get the handles to everything that's enabled
    set([handles.LoadSettings,handles.HardwareMenu],'Enable','off')%turn off a couple of menu options
    hm = findobj(handles.figure1,'-property','Checked','Enable','on'); %get the handles to all menu options that are
enabled
    on = [hm;handles.CCfit;handles.PauseScan;handles.ExitScan]; %we want to turn on all menus and some controls
    set(ha,'Enable','off'), set(on,'Enable','on')   %turn everything either off or on
    vis = handles.MainBack;
    if ~handles.value.backsub, vis=[vis handles.PumpBack handles.ProbeBack]; imagenum=imagenum*3; end
    set(vis,'Visible', 'on'),drawnow                %let the user see the inensity vs time graph(s)

% Set up a filename
    name = [Date '-' num2str(Count,'%02d')];
    path = getpath;      %get the file path name
    [name, path, fileok]=uiputfile([path name '.mat'],'Select filename and location for image files');
    if fileok==0                                    %if shutter not working, exit
        ExitScan=1;ferr='No file chosen';
        errordlg(ferr,' File Error'),uiwait
```

```
    else name=name(1:end-4);                        %remove the '.mat' from the end
    end
    waitbar(0.2);

%Setup Shutters:
    if ExitScan==0,err=setupshutters(handles);end
    %if there is an error, check if you need shutters
    if ~isempty(err), sherr=1;
        if  get(handles.backsub,'Value'),  helpdlg([err  sprintf('\n\nPlease  make  sure  shutters  are  manually
opened')],'Shutter Error'),uiwait
        else ExitScan=1;helpdlg([err sprintf( '\n\nTo run scan, please set Background Subtraction to off and open
shutters manually. Scan will now terminate.')],'Shutter Error'),uiwait,end
    else err=ferr;
    end
    waitbar(0.4);

%Setup Stage part 2 (check how stage referincing is getting on. If it's
%still moving wait a bit longer, then move to defined home)
    if ExitScan==0
        err=setupstage(handles,2);                          %setup stage part 2
        %if stage was working but has failed, close connection and try again.
        if err~=0 && ~isempty(stage)
            CloseConnection(stage), stage=[];pause(0.5);%close connection
            err=setupstage(handles,1);%do initial setup again
        end
    end
    waitbar(0.7);
%Setup Camera:
    if ExitScan==0
        try
            set(handles.textdisplay,'String','setting up camera...')
            vid = videoinput('winvideo',1);             %define camera
            src=getselectedsource(vid);
            frame=set(src,'FrameRate');                 %Sets the framerate.
            src.FrameRate = frame{handles.value.framerate};
            camcontrast=set(src,'ContrastMode');        %Sets the contrast mode to
            src.ContrastMode = camcontrast{2};          %manual({1}=auto,{2}=manual)
            src.Contrast = handles.value.camcontrast;   %and the contrast value.
            triggerconfig(vid, 'Manual')
            set(vid,'TriggerRepeat',Inf)
%             set(vid,'FramesPerTrigger',max(shotspp,shotsb,shotsc))%how many pictures for each image
%             set(vid,'TriggerFrameDelay',5);             %get rid of the first 5 pictures (camera warmup)
            set(handles.figure1,'BackingStore', 'off'); %increases drawing speed
        catch                                           %if camera not working, exit
            error=lasterror;ExitScan=1;
            err=['Camera error: ' error.message];
            errordlg([err 'Try checking the cables.'],' Camera Error'), uiwait
        end
    end
    waitbar(0.9);
%check how stage is getting on again. If it's still moving wait a bit longer, then set home to zero.
    if ExitScan==0
        stagemove(stage,stageaxis,handles,0)
        DFH(stage,stageaxis)                                %set new "Home"
        set(handles.stageslider,'Min',qTMN(stage,stageaxis))%set up slider.
        set(handles.stageslider,'Max',qTMX(stage,stageaxis))
        MOV(stage,stageaxis,dsign*delays(1)*0.0002998);     %Move the stage to the first position
    end
    waitbar(1);
    csatv=zeros(1,shotspp);
    filename = [path name 'settings.mat'];                  %set filename
    %picdisp(tdata,handles,10,filename,0);                  %set up all the graphs point
    try close(barh),catch ExitScan=1;end
    bg=10;is=0;sccomp=0;
    tic                                                     %start timer running (for calculating time left)
end
rdata=zeros(480,640,1,2);
posv=dsign*delays*0.0002998;  posv(end+1)=posv(1);          %position vector
```

```
%% 'Run' loop: Controls the equipment, gets the data and saves it
%loop for each scan
for scan = sc:handles.value.scannum
    if ExitScan==1, filename='Scan Terminated. '; break, %Exit Run if 'Abort Run' has been pressed or error has occured
    elseif ExitScan==2, %Exit Run if 'Pause Run' has been pressed
        set(handles.gaustxt,'string','Scan Paused. Press Pause again to restart. ');
        set(handles.PauseScan,  'Enable',  'on'),break,
    end
    sccomp=scan-1;
    flushdata(vid),iset=1;                          %clear memory & reset picture no. to 0
    stagemove(stage,stageaxis,handles,1)
    MOV(stage,stageaxis,posv(1));                    %Move the stage
%loop for each picture
    for pic = 1:imagenum
        if ExitScan==1,filename='Scan Terminated. ';break,end%Stop if 'Abort Run' has been pressed or error has occured
        try
            %Setup picture specific variables and operate shutters
            if ~sherr,shutterop(handles,1,1);shutterop(handles,2,1);end %open shutters
            if handles.value.backsub||rem(pic,3)==0     %main picture
                iset=iset+1; bkg=1; fframes=shotspp;
            elseif rem(pic+2,3)==0                      %pump alone
                if ~sherr,shutterop(handles,1,0);end    %close shutter 1 (probe shuter)
                bkg=2; fframes=shotsb;
            elseif rem(pic+1,3)==0                      %probe alone
                if ~sherr,shutterop(handles,2,0);end    %close shutter 2 (pump shuter)
                bkg=3; fframes=shotsc;
            end
            set(vid,'FramesPerTrigger',fframes)     %how many pictures for each image
            if isrunning(vid),stop(vid),end         %check the camera is good
            start(vid)                              %turn on camera
            stagemove(stage,stageaxis,handles,1);   %wait for stage to stop moving & display position as it moves
            trigger(vid)                            %start aquiring images
            %process last set of data while new pic is being taken
            for r=1:size(rdata,4)
                picc=rdata(:,:,:,r);
                picc(picc<=imthresh)=0;             %remove camera noise
                rdata(:,:,:,r)=picc;
                csatv(r)=any(any(picc>=255));
            end
            csat    = any(csatv);                   %if camera is saturating, throw a warning
            tdata   = sum(rdata,4);                 %add them together
            tdata(178,151)=0;                       %get rid of dodgy pixel
            picdisp(tdata,guidata(gcbo),bg,filename,is);  %display/save the old picture
            bg=bkg;is=iset;                         %set saving values for next picture
            filename= [path name '-' num2str(scan) '.mat'];
            wt=(pic+imagenum*(scan-1))/(imagenum*handles.value.scannum);
            %update the "time left" text
            timestring=['Date: ' date '.  Approx: ' timedisp(toc/wt-toc) ' left till Run completed.\n'...
                'Scan has been running for ' timedisp(toc) '. Scan ' num2str(scan) ' of ' num2str(handles.value.scannum)];
            set(handles.displaydate, 'String',sprintf(timestring))
            while islogging(vid),pause(0.05),end    %wait till camera has finished aquiring image
            MOV(stage,stageaxis,posv(iset));            %Move the stage to next position (dsign=sign of delays, delays=list of stage positions in fs, iset=index of particular position this time, 0.0002998=scaling factor from fs to um (if you're scanning backwards dsign=-1, otherwise 1))
            rdata = getdata(vid,fframes);           %get all the frames
            flushdata(vid), stop(vid)               %clean up camera
        catch                                       %if an error occurs
            error=lasterror;error.stack.line,err=error.message %get the error and location of error and write it to the matlab window
            ExitScan=1; break                       %exit the Scan
        end %stop looking for errors
    end %end picture loop
end %end scan loop
%process final picture
```

```
for r=1:size(rdata,4)
    picc=rdata(:,:,:,r);
    picc(picc<=imthresh)=0;             %remove camera noise
    rdata(:,:,:,r)=picc;
    csatv(r)=any(any(picc>=255));
end
csat    = any(csatv);                   %if camera is saturating, throw a warning
tdata   = sum(rdata,4);                 %add them together

if ExitScan==1 && pic~=1                            %if Run Aborted by User mid scan
else picdisp(tdata,guidata(gcbo),bg,filename,is);   %display/save the final picture.
end

catch %check for any errors not already caught
    filename='Scan Terminated. '; errrr=lasterror;
    errrr.stack.line,err=errrr.message               %get the error and location of error and write it to the matlab
window
end
% set(,'Enable','on')
set([ha;handles.PauseScan],'Enable' ,'on')
if ExitScan==2 && scan<handles.value.scannum        %if Run is paused, don't close hardware or tidy up.
else
    %Tidy up
    ExitScan=0; Running=0; c=clock;                 %reset ExitScan & Running & get end time
    %deal with hardware
    stagemove(stage,stageaxis,handles,1)            %wait for stage to stop moving & display position as it moves
    GOH(stage,stageaxis),                           %set stage to home position
    if ~sherr,shutterop(handles,1,1),shutterop(handles,2,1),end %open shutters
    try flushdata(vid),delete(vid), end             %stop camera
    %Save Random Stuff (Data files already saved by picdisp)
    try load([path name '~1.mat']), Count=Count+1;  %if no scans saved, delete settings file and re-use run #
    catch
        savef ='No';
        if pic                                      %if there's anything to save
            savef = questdlg('Would you like to save the incomplete scan? (all complete scans in this run have been
saved)','Abort Run','Yes','No',savef);
        end
        if strcmp(savef,'Yes'), picdisp(tdata,handles,9,[path name '-' num2str(scan) '.mat'],is)%save values
        else delete([path name 'settings.mat']); filename='File not Saved. ';end %don't save values and re-use
filename for next scan
    end
    save('name','Date','Count'),save('dataf','data','wd'),fclose('all') %Save variables
    %User Interface
    off = [handles.ExitScan,handles.PauseScan];
    set(off,'Enable' ,'off')
    set(handles.runcounter,'String', Count);        %update runcounter
    finstring=['Date: ' date '. Run finished at ' num2str(c(4)) ':' num2str(c(5)) ...
        ' after completing ' num2str(sccomp) ' scans.\nRun took ' timedisp(toc) ' to complete'];
    set(handles.displaydate, 'String', sprintf(finstring));  %display finishing time and stats
    set(handles.textdisplay, 'String', ['Finished: ' filename err])  %display filename
    guidata(hObject,handles)
end
if cl==5,cl=4;close,end                             %find out if the program is trying to close
stagemove(stage,stageaxis,handles,1)                %wait for stage to stop moving & display position as it moves
%% 'Live Camera View' Button
function campreview_Callback(hObject, eventdata, handles)
global vid snapshot datap
settings=[handles.CameraSettings,handles.StageSettings,handles.AquireSettings];
sidegraphs=[handles.Spectra,handles.Delays,handles.ProbeBack,handles.PumpBack,handles.MainBack];
snapshot=0;
if get(handles.campreview,'Value')  %if "preview" is pressed down
    vid = videoinput('winvideo',1); %set up camera
    %setup user interface
    cla(handles.Delays,'reset'), cla(handles.Spectra)%take lines off sidegraphs
    set(handles.Run,           'Enable', 'off')
    set(handles.PauseP,        'Enable', 'on' )
    set([settings,sidegraphs], 'Visible','off')
    set(handles.PauseP,        'Value'  , 0   )
```

```
    campreview(hObject, handles) %go to preview function
else%if "preview" is pressed up
    load dataf,
    stoppreview(vid), closepreview(vid), flushdata(vid)%stop preview function
    % tidy up
    data=datap/max(max(datap))*255;
    save('dataf','data','wd');
    picdisp1(data,handles,['Aquired for ' num2str(round(toc)) ' s. ']);
    set(handles.Run,     'Enable', 'on' )
    set(handles.PauseP,  'Enable', 'off')
    set(settings,        'Visible','on' )
    cla(handles.IntenTrace)
    set(handles.IntenTrace,'Visible','off')
    load piccentre
    handles.value.piccentre=cn;
    handles.value.picwidth=wd;
    guidata(hObject,handles)
end

% set up preview function
function campreview(hObject, handles)
global vid trace b stage mask mi wd cn
b=0;
if ~isempty(stage),stagepos(handles),end
if get(handles.campreview,'Value') && ~get(handles.PauseP,'Value')
    %set up camera
    flushdata(vid)
    src=getselectedsource(vid);
    frame=set(src,'FrameRate');                    %Sets the framerate.
    src.FrameRate = frame{handles.value.framerate};
    camcontrast=set(src,'ContrastMode');           %Sets the contrast mode to
    src.ContrastMode = camcontrast{2};             %manual({1}=auto,{2}=manual)
    src.Contrast = handles.value.camcontrast;      %and the contrast value.
    %get the data and plot it
    axes(handles.IntenTrace),hIntenGrph = plot(trace,'YDataSource','trace');%set up average intensity graph
    axes(handles.Picture), hImage = image(zeros(480,640));axis image %set up preview himage
    setappdata(hImage,'UpdatePreviewWindowFcn',@mypreview); %Set up the update preview function.
    setappdata(hImage,'IntenHandle', hIntenGrph);   %Make intensity graph available to mypreview function.
    setappdata(hImage,'AquireHandle',handles.Graph);%Make aquired image axis available to mypreview function.
    setappdata(hImage,'TextHandle',handles.textdisplay) %Make text available to mypreview function.
    setappdata(hImage,'CSat',handles.csat)         %Camera Saturated Button available
    load piccentre;   mi; wd; cn;                  %set up mask & crop variables
    [x,y]=ndgrid((1:480)-cn(2),(1:640)-cn(1)); mask=(x.^2+y.^2)<mw^2;%calculate mask
    preview(vid,hImage)                            %run preview(each time it gets a new image, mypreview runs)
else stoppreview(vid), closepreview(vid),flushdata(vid)%stop preview function
end

%  mypreview function (executes with every image)
function mypreview(obj,event,himage)
global trace numav b cropt datap prevcontrast time imthresh livethresh...
    maskc maskt cropd cn wd mi mask centon crson symon
b=b+1;                                             %loop count
event.Data(178,151)=0;                             %get rid of dodgy pixel
aquiredata = double(event.Data);                   %get image
aquiredata = centroid(aquiredata,imthresh,centon);
% aquiredata(aquiredata<=imthresh)=0;              %remove camarea noise
datap      = datap+aquiredata;                     %add image to aquired image
l          = size(trace,2);
if l<50,beg= 1; else beg = l-48; end
if maskt,a=sum(sum(aquiredata(mask)))/(numav*640*480);%get the average intensity of centre of image
else a=mean(mean(aquiredata))/numav; end           %get the average intensity of whole image

if b==1;                                           %first time through
    if numav==1,trace(end)=a; else trace(end)=2*a; end
elseif rem(b,numav)==0                                %every mth picture add an intensity point onto the average
intensity plot
    graph = getappdata(himage,'IntenHandle');      %get axis to plot in
    txt   = getappdata(himage,'TextHandle');       %get text handle to write to
```

```
    refreshdata(graph,'caller')                    %refresh intensity graph
    set(txt,'String',['Intensity = ' num2str(mean(trace(beg:end)),3) ...
        ' averaged over ' num2str(l-beg+1) ' points'])
    trace=[trace a];                               %create the intensity graph vector
else trace(end)=trace(end)+a;                      %add the intensities
end
if rem(b,15)==0 || b==1                                %every 15th pic update the aquired image pic (note:15 isn't a
magic no. I used it becuse my computer was too slow to handle updating with every image)
    data=datap;                                    %make dataset that it doesn't matter if you modify
    if maskc,    data(mask)=data(mask)*mi;end      %reduce intensity of centre portion
    if symon,    data=symetrise(data,cn(2),cn(1),wd);                    %symatrise picture
    elseif cropd,data=data(cn(2)-wd+1:cn(2)+wd,cn(1)-wd+1:cn(1)+wd);end  %crop image
    if crson,    data(:,wd-1:wd+1)=0; data(wd-1:wd+1,:)=0;end            %make centre cross
    aqim  = getappdata(himage,'AquireHandle');                           %get axis to plot in
    imshow(data/max(max(data))*prevcontrast*10,colormap(jet),'Parent',aqim)  %plot aquired image (feel free to play
with multiplying factors to get nice colormap)
    minutes='';hours='';seconds=[num2str(round(rem(toc,60))) ' seconds.'];  %make time string I
    if toc/60>1,minutes=[num2str(floor(rem(toc/60,60))) ' minutes ' ];end   %make time string II
    if toc/3600>1,hours=[num2str(floor(toc/3600)) ' hours ' ];end           %make time string III
    time=['Image Aquired for: ' hours minutes seconds ];                    %make time string IV
    xlabel(aqim,time),title(aqim,'Aquired Image')                          %annotate graph
    if max(max(aquiredata))>=200, set(getappdata(himage,'CSat'),'Visible','on'),end %check that you're not saturating
the camera
end

if l > 99 && cropt %keep trace short if required
    trace=trace(end-99:end);
end
if livethresh, event.Data=aquiredata; end
set(himage, 'CData', event.Data*prevcontrast)    %Display image data.
drawnow
%% Live View Sectionbox

% Pause Live View button
function PauseP_Callback(hObject, eventdata, handles)
campreview(hObject, handles)

% Preview Contrast Slider
function     prevcontrast_CreateFcn(hObject,     eventdata,     handles),if     isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function prevcontrast_Callback(hObject, eventdata, handles)
global prevcontrast, prevcontrast=30*get(handles.prevcontrast,'Value');
set(handles.conttxt,'String',num2str(prevcontrast,'%2.1f')) %display new value

% Reset Aquired Image button
function aqreset_Callback(hObject, eventdata, handles)
global datap, datap=0; tic %reset datap, reset timer

%No of pics to average over editbox
function    numav_CreateFcn(hObject,    eventdata,    handles),if    ispc    &&    isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function numav_Callback(hObject, eventdata, handles)
global numav, numav=edit(hObject,numav,0,Inf,1,'Number of Pictures');

%Tickboxes
function livethresh_Callback(hObject, eventdata, handles)
global livethresh, livethresh=get(hObject,'Value'); %Show Threshold Image Tickbox
function cropd_Callback(hObject, eventdata, handles)
global cropd, cropd=get(hObject,'Value');           %Crop Aquired Image Tickbox
function preInten_Callback(hObject, eventdata, handles)
global cropt, cropt=get(hObject,'Value');           %Crop Bottom Trace Tickbox
function maskt_Callback(hObject, eventdata, handles)
global maskt, maskt=get(hObject,'Value');           %Average Over Mask Area Tick Box

% Modify button (openes a new window)
function aqsave_Callback(hObject, eventdata, handles)
global datap prevcontrast time saveno maskc cropd
%Open new window
```

```
saveno=saveno+1; num=num2str(rem(saveno,100));      %store data in uneque place
col=get(gcf,'Color');                               %make the new window and buttons the same background colour
h=figure('Name',time,'Position',[80 60 800 600],'Color',col); %make new window
%setup variables
timestamp=[time ' Saved on ' date];                 %string displaying time aquired and date saved
data=datap;                                         %get latest image data
load piccentre                                      %load centrepoint and masking conditions
c.ud=cn(1);c.lr=cn(2);c.w=wd;                       %save centrepoint and width
c.mw=mw;   c.mi=mi;                                 %save mask width and intensitry
[x x im] = getpath;                                %get the file path name

%Set Up and run figure User Interface
    %create an axis to draw in
axes('Position',[0.1 0.16 0.8 0.8],'Parent',h);
%Buttons:
    %Save Data Button (Saves image.mat to new, user defined, location)
uicontrol(h,'Style', 'pushbutton', 'String', 'Save Data','Position', [80 50 80 25],...
     'Callback',['load im\image' num '.mat,uisave({''data'',''sdata'',''timestamp'',''c''},''' im '*.mat '')']);
    %Save Picture Button (Saves figure window (witout UI controls) to tif image)
uicontrol(h,'Style', 'pushbutton', 'String', 'Save Picture','Position', [180 50 80 25],...
     'Callback',['[name,path]=uiputfile(''' im '*.tif'');print(''-noui'',''-dtiffn'',[path name])']);
    %Save piccentre button
uicontrol(h,'Style', 'pushbutton', 'String', 'Save Centre Point and Masking Conditions','Position', [80 5 280 25],...
     'Callback',['global cn wd mw mi mask,load im\image' num '.mat,cn=[c.ud c.lr];'...
     'wd=c.w;mw=c.mw;mi=c.mi;save piccentre cn wd mw mi;[x,y]=ndgrid((1:480)-c.lr,'...
     '(1:640)-c.ud); mask=(x.^2+y.^2)<c.mw^2;']);
    %Crop Button (Crops image to same size and centrepoint as symetrised data uses)
cr=uicontrol(h,'Style', 'togglebutton', 'String', 'Crop','Position', [280 50 80 25],'Value',cropd);
    %Symetrise Button (symetrises data around centrepoint defined in edit boxes)
sy=uicontrol(h,'Style', 'togglebutton', 'String', 'Symetrise','Position', [640 50 80 25]);
    %Mask button (lowers intensity around centre point, so dimmer outer rings can be observed)
mk=uicontrol(h,'Style', 'togglebutton', 'String', 'Mask Centre','Position', [375 5 80 25],'Value',maskc);
    %plot function (works out what buttons are pressed, and updates image to appropriate data)
plotsym=['try,sdata=symetrise(data,c.lr,c.ud,c.w);'...              %symetrise data around centre point
        'save im\image' num '.mat data sdata timestamp c,'...           %save all data
        '[x,y]=ndgrid((1:480)-c.lr,(1:640)-c.ud); mask=(x.^2+y.^2)<c.mw^2;'... %calculate mask
        'if get(' num2str(mk,100) ','''Value''),data(mask)=data(mask)*c.mi;'... %check if 'Mask' is pressed, and mask
data if required
        'sdata=symetrise(data,c.lr,c.ud,c.w);end,'...               %symetrise with mask (but don't save)
        'if  get('  num2str(cr,100)  ','''Value''),data=data(c.lr-c.w+1:c.lr+c.w,c.ud-c.w+1:c.ud+c.w);end,'...%check  if
'crop' is pressed, and crop data if required
        'if  get('  num2str(sy,100)  ','''Value''),data=sdata;end,'...   %check  if  'symetrise'  is  pressed,  and  used
symetrised data if required
        'imshow(data/max(max(data))*' num2str(prevcontrast) '*10,colormap(jet)),'...%plot appropriate image
        'catch,err=lasterror,err.stack.line,errordlg([''please  check  your  values.  ''  err.message]),end'];%deal  with
errors
    %tell buttons to use plot function
set(cr,'Callback',['load im\image' num '.mat,' plotsym]);             %make crop button run plotsym when pressed
set(sy,'Callback',['load im\image' num '.mat,' plotsym]);             %make sym  button run plotsym when pressed
set(mk,'Callback',['load im\image' num '.mat,' plotsym]);             %make mask button run plotsym when pressed
%Editboxes (retrieves new value then runs plotsym which saves new value and updates image)
    %Width Edit Box
uicontrol(h,'Style', 'edit', 'String', num2str(c.w*2),'Position', [380 50 70 25],'Background','white',...
     'Callback',['load im\image' num '.mat,c.w=str2double(get(gcbo,''String''))/2;' plotsym]);
uicontrol(h,'Style', 'text', 'String', 'Width:','Position', [380 75 70 15],'Background',col)%Width Text
    %Centrepoints Edit Boxes
uicontrol(h,'Style', 'edit', 'String', num2str(c.ud),'Position', [470 50 70 25],'Background','white',...
     'Callback',['load im\image' num '.mat,c.ud=str2double(get(gcbo,''String''));' plotsym]);
uicontrol(h,'Style', 'edit', 'String', num2str(c.lr),'Position', [550 50 70 25],'Background','white',...
     'Callback',['load im\image' num '.mat,c.lr=str2double(get(gcbo,''String''));' plotsym]);
uicontrol(h,'Style', 'text', 'String', 'Centrepoint:','Position', [470 75 150 15],'Background',col)%Centrepoint Text
    %Mask Radius Edit Box
uicontrol(h,'Style', 'edit', 'String', num2str(c.mw),'Position', [550 5 70 25],'Background','white',...
     'Callback',['load im\image' num '.mat,c.mw=str2double(get(gcbo,''String''));' plotsym]);
uicontrol(h,'Style', 'text', 'String', 'Mask Radius:','Position', [550 30 70 15],'Background',col)%Width Text
    %Mask Intensity Edit Box
uicontrol(h,'Style', 'edit', 'String', num2str(c.mi),'Position', [470 5 70 25],'Background','white',...
     'Callback',['load im\image' num '.mat,c.mi=str2double(get(gcbo,''String''));' plotsym]);
```

```
uicontrol(h,'Style', 'text', 'String', 'Mask Intensity:','Position', [470 30 70 15],'Background',col)%Width Text
%Save Variables and Display Image when Figure is first Opened
eval(plotsym);                                                  %run plot function

%% '
%% GENERAL FUNCTIONS
%time display function (takes a time in seconds and returns a string saying
%time in days, hours, minutes and seconds
function string=timedisp(toctime)
if toctime<180, s=1;else s=0;end                                %decide whether to display seconds
d=floor(toctime/86400);h=floor(toctime/3600);m=floor(toctime/60);
day='';hour='';minute='';second='';
if d>0,     day   =[num2str(d,'%-3.0f') ' days, '];end          %days
if h>0,     hour  =[num2str(rem(h,24),'%-3.0f') ' hours, '];end  %hours
if s,       second=[num2str(rem(toctime,60),'%-3.0f') ' seconds'];  %seconds
    if m>0, minute=[num2str(rem(m,60),'%-3.0f') ' minutes and '];end%minutes
else        minute=[num2str(rem(m,60),'%-3.0f') ' minutes'];
    if ~isempty(hour), minute=['and ' minute]; end                  %put the 'and' before minutes if it's the last
thing
end
string=[day hour minute second];                                %complete string

%centoid function
function x=centroid(data,thresh,centon)
data(data<thresh)=0;
if centon
    base=zeros(size(data,1)+2,size(data,2)+2,size(data,3));
    base(2:end-1, 2:end-1, :)=data;
    %see which pixels work
    x1 = data >  base(1:end-2,2:end-1,:);   %greater than the pixel on the left
    x2 = data >= base(3:end,2:end-1,:);     %greater than the pixel on the right
    x3 = data >  base(2:end-1,1:end-2,:);   %greater than the pixel above
    x4 = data >= base(2:end-1,3:end,:);     %greater than the pixel below
    x  = x1 & x2 & x3 & x4;                 %greater than each of these pixels
else x=data;
end
x=sum(x,3);
%% Set up Stage
function err=setupstage(handles,location)
%location: 0=Run(start); 1=timinghelp section, 2=Run(end)
global stage stageaxis ExitScan startup
err=0; c=''; b=60; startup=0; error='';
if isempty(stage) && location<2 %only do this if there isn't a stage already
    startup=1;
    set(handles.textdisplay,'String','setting up stage...')
    stage = Mercury_Controller();
    stage = InterfaceSetupDlg(stage);
    stage = InitializeController(stage);
    stageaxis = qSAI_ALL(stage);                %talk to stage controller
    CST(stage,stageaxis,'M-403.62S')            %connect to stage
    INI(stage,stageaxis);                       %initialise stage
    REF(stage,stageaxis);                       %move stage till it trips a reference switch(so it knows where it is)
    err=qERR(stage);                            %get any errors
    if err==1,
        errordlg('Stage not working. Get stage working again by unplugging USB cord breafly')
        stage='';ExitScan=1;uiwait,return
    elseif ~IsConnected(stage)
        error=['Stage not connected: ' TranslateError(stage,err)];                  %if it's not working, exit
        ExitScan=1;
    else set(handles.textdisplay,'String','stage moving...')
    end
end

if IsConnected(stage) && location>0&& err==0 %don't do this at the start of Run setup
    while stage.isreferencing(stageaxis)%wait till stage stops moving
         pause(0.5),b=b-1;set(handles.stagepostxt,'String',['wait...' num2str(b,'%11.0f')])
    end
    hom=qTMN(stage,stageaxis);
```

```
    if handles.value.home ~=hom;
        MOV(stage,stageaxis,-handles.value.home+hom)
        err=qERR(stage);
        TranslateError(stage,err) %display on Matlab Screen
        if err==0&&location==1,stagemove(stage,stageaxis,handles,0),end %don't wait for "Run" function
        if err~=0,
            error=['Stage error: ' TranslateError(stage,err)];ExitScan=1;
        elseif location==1, DFH(stage,stageaxis);    %set new home value
        end
    end
end
set(handles.textdisplay,'String',error)
%% Set up & Operate shutters
%set up shutters
function err=setupshutters(handles)
err='';
try
    if ~handles.value.shutters%if shutters haven't been initialised
        handles.activex2.StartCtrl; %set up shutter 1
        handles.activex3.StartCtrl; %set up shutter 2
    end

    sh1=handles.activex2.SC_SetOperatingMode(0,1);
    sh2=handles.activex3.SC_SetOperatingMode(0,1);

    % SetOperatingMode(channel,mode) Channel=0. Mode=:{1(on/off); 2(on for
    % a set time, then off); 3(continuasly on,off,on,off..); or 4(trigger)}

    if sh1 && sh2,
        err='Both shutters are not working properly.';
        handles.value.shutters=0; guidata(gcbo,handles)
    elseif sh2
        err='Shutter 2 is not working properly.';
        set(handles.shutter1,'Enable','on');
        shutterop(handles,1,1)
    elseif sh1,
        err='Shutter 1 is not working properly.';
        set(handles.shutter2,'Enable','on');
        shutterop(handles,2,1)
    else
        set(handles.shutter1,'Enable','on');
        set(handles.shutter2,'Enable','on');
        shutterop(handles,1,1),shutterop(handles,2,1)
        handles.value.shutters=1;guidata(gcbo,handles)
    end
catch
    error=lasterror; err=error.message
end


%operate shutters (Opens and Close)
function shutterop(handles,shutter,open)%shutter = 1 OR 2,open = 0 OR 1
if shutter==1;
    if open,handles.activex2.SC_Enable(0);      %open  shutter 1
    else    handles.activex2.SC_Disable(0);end  %close shutter 1
    h = handles.shutter1;
else
    if open,handles.activex3.SC_Enable(0);      %open  shutter 2
    else    handles.activex3.SC_Disable(0);end  %close shutter 2
    h = handles.shutter2;
end
%change the tooltip string of the shutter buttons
if open,string= ' Close Shutter ';
else    string= ' Open Shutter '; end
set(h,'TooltipString',string)
%change the toggle status of the shutter buttons
set(h,'Value',open)
%% Run Counter for filenames
```

```
function runcounter_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end %set background colour
function runcounter_Callback(hObject, eventdata, handles)
load name
uservalue = str2double(get(hObject, 'String'));
if uservalue ~= Count
    h=questdlg('Are you sure you want to change the counter? Previous files may be overwritten');
    if strcmp(h,'Yes')
        if isnan(uservalue),errordlg('Input must be a number','Error');
        else Count = uservalue;end
    end
end
% Save the new runcounter value
    save('name','Date','Count')
    handles.value.runcounter = Count; guidata(hObject,handles)

%% Close figure
function figure1_CloseRequestFcn(hObject, eventdata, handles)
global Running stage ExitScan cl, h='';

if Running %if run is on, close hardware safely first, then close window
    h=questdlg('Are you sure you want to close the program and exit the scan? If you press "No" the scan will continue
uninterrupted.');
    if strcmp(h,'Yes')
        load name,ExitScan=1; %Abort run safely
        save('name','Date','Count')
    end
else %if not running
    if ~isempty(stage), CloseConnection(stage), stage=[]; end   %stop stage
    if handles.value.shutters, handles.activex2.StopCtrl; handles.activex3.StopCtrl;end %stop shutters
    delete(hObject) %close figure.
end
if Running && strcmp(h,'Yes'), cl=5; %once run is Aborted, come back here
else cl=0;end

%% Calculate Number of Images
function imagenum_Calculate(hObject, handles)
stagestart = handles.value.stagestart;
stagestep  = handles.value.stagestep;
stageend   = handles.value.stageend;
expstageend= handles.value.expstageend;
exppoints  = handles.value.exppoints;
delays1    = stagestart:stagestep:stageend;
if exppoints==0 || expstageend<=stageend, delays2=[];
else d=logspace(log10(stageend),log10(expstageend),exppoints+1); delays2=d(2:end);end
handles.value.delays = [delays1 delays2];
delaysl = length(handles.value.delays);
% Save the new delays
guidata(handles.imagenum,handles)
set(handles.imagenum,'String', num2str(delaysl));

%% Picture Function (creates picture and graph and saves data)
function picdisp(rdata,handles,setting,textdisplay,iset)
global spec delays rect scan imthresh csat sccomp api centon
persistent hI hG Pe Pm Pp Pd Ps PumpPlot ProbePlot PumpProbe Dataa Datab Datac x0 y0 xx yy r a %#ok<PUSE>
%rdata      = raw data to be made into cropped picture and graph
%setting    = Value to change output:
%                  -1 (gaussian fit button)
%                   1 (Main Picture)
%                   2 (Pump only Picture),or
%                   3 (Probe only Picture)
%                   9 (Run Aborted)
%                  10 (Set up pics for first time)
%textdisplay= string to display in window
%handles    = all the variables and values
%get and calculate variables
x     = handles.value.piccentre(1);            %x value of centre
y     = handles.value.piccentre(2);            %y value of centre
```

```
w     = handles.value.picwidth;                 %width is the shortest distance from the centre to the edge
cdata = rdata(y-w+1:y+w,x-w+1:x+w);             %crop picture
cdata = centroid(cdata,imthresh,centon);        %subtract camera noise and centroid image, as requested
inten = mean(mean(cdata(:,:)));                 %get the average intensity
%set up variables to be saved
sett2 = iset+length(delays)*sccomp;
%Create and display Graphs:
if setting==10                                  %Initial Set up
%Set up variables
    shotspp= round(handles.value.imagetime*120/(2^handles.value.framerate));if shotspp==0, shotspp=1; end
    shotsb = round(handles.value.btime*120/(2^handles.value.framerate)); if shotsb==0, shotsb=1; end
    shotsc = round(handles.value.ctime*120/(2^handles.value.framerate)); if shotsc==0, shotsc=1; end
    if handles.value.backsub, shotsb=0; shotsc=0;end
    i     = size(cdata);
    Dataa = zeros(i(1),i(2),length(delays),'int8');%reset everything to zero and asign space to large variables
    Datab = Dataa; Datac = Dataa;
    [x0 y0] = meshgrid([1:2*w]-w,[1:2*w]-w);        %setup variables for integrating in a circle
    [r  a ] = meshgrid(1:w,linspace(0,2*pi,180));
    [xx yy] = pol2cart(a,r);
%SET UP GRAPHS
%set up spectra/delays graph
    axes(handles.Graph)
    spec= zeros(length(delays),w);     %reset values
    hG  = surf(1:w,delays,spec,spec,'CDataSource','spec');%plot Graph
    view(handles.Graph,0,90),shading flat,colormap jet, axis tight
    rect= [1 delays(1) w-2 delays(end)-delays(1)];             %set up selection rectangle
    h2  = imrect(handles.Graph,rect);
    api = getappdata(h2,'API');
    fcn = makeConstrainToRectFcn('imrect',get(handles.Graph,'XLim'),get(handles.Graph,'YLim'));
    api.setDragConstraintFcn(fcn);
    api.addNewPositionCallback(@RectFn)
%set up camera image graph
    axes(handles.Picture);
    hI=surf(1:i(2),1:i(1),zeros(i(1),i(2)),cdata,'CDataSource','cdata');%Display the picture
    shading interp;axis equal tight off;view(handles.Picture,0,90)
%set up the background intensity graphs
    a=zeros(1,length(delays)*handles.value.scannum);
    PumpPlot=a; ProbePlot=a; PumpProbe=a; b=1:length(a);
    axes(handles.ProbePlot), Pe=plot(b,ProbePlot,'YDataSource','ProbePlot'); axis off
    axes(handles.PumpPlot),  Pm=plot(b,PumpPlot ,'YDataSource','PumpPlot' ); axis off
    axes(handles.PumpProbe), Pp=plot(b,PumpProbe,'YDataSource','PumpProbe'); axis off
%set up delays and spectra sidegraphs
    cla (handles.Delays)
    axes(handles.Delays),    Pd=plot(delays,delays,'XDataSource','mean(trimspec,2)','YDataSource','x');
    axes(handles.Spectra),   Ps=plot(1:w,1:w,'XDataSource','rw1:rw2','YDataSource','mean(trimspec,1)');
    ylim(handles.Delays, [delays(1) delays(end)]), xlim(handles.Spectra,[0 w])
    set(Pd,'HandleVisibility','off')                %this is so I can plot gaussian fit later
%Save settings file
    Isize=i; scannum = handles.value.scannum;
    save(textdisplay,'delays','shotspp','shotsb','shotsc','scannum','Isize')
elseif setting==1||setting==-1                  %Main Picture
    if setting==1
        rInt = interp2(x0,y0,cdata,xx,yy);       %turn cartesian image into polar image
        spec(iset-1,:)        = spec(iset-1,:)+sum(rInt,1);
        PumpProbe(sett2-1:end)= inten;
        Dataa(:,:,iset-1)     = int8(ceil(cdata)-128); %setup data for file
        refreshdata(hI,'caller')                %display Image
    end
    rect=api.getPosition();                     %find out where the draggable rectangle is
    r1=find(delays<rect(2));r2=find(delays>rect(4)+rect(2));%put that in context of the graph
    if isempty(r1),rdel1=1;else rdel1=r1(end)+1;end
    if isempty(r2),rdel2=length(delays);else rdel2=r2(1)-1;end
    x=delays(rdel1:rdel2)';
    rw1=round(rect(1));rw2=round(rect(3)+rect(1));
    trimspec=spec(rdel1:rdel2,rw1:rw2);         %crop the sidegraph data
    refreshdata(hG,'caller')                    %plot Graph
    refreshdata(Pp,'caller')                    %plot intensity
    refreshdata(Ps,'caller')                    %plot spectra sidegraph
```

```
    refreshdata(Pd,'caller')                    %plot delays sidegraph
    if setting==-1 || (iset-2)==(length(delays)-1)%only fit crosscorelation if end of scan
        y = tracefit(x,mean(trimspec,2),handles);%fit crosscorelation
        if ~isempty(y)
            axes(handles.Delays),cla(gca)       %clear last fit line
            refreshdata(Pd,'caller'),hold on
            plot(handles.Delays,y,x,'-r'),hold off %plot new fit line
        end
    end
    if (iset-1)==length(delays)
        disp(['scan ' num2str(scan-1) ' finished at ' datestr(now,16) ', saved as: ' textdisplay])
        save(textdisplay,'Dataa','Datab','Datac')%SAVE SCAN DATA.
    end
    if csat,camerasat(handles,textdisplay,scan,iset),end
elseif setting==2                               %Pump only Picture
    PumpPlot(sett2:end)=inten;                  %add data to pump background plot vector
    refreshdata(Pm,'caller')                    %plot pump background graph
    Datab(:,:,iset)=int8(ceil(cdata)-128);      %setup data for file
elseif setting==3                               %Probe only Picture
    ProbePlot(sett2:end)=inten;                 %add data to probe background plot vector
    refreshdata(Pe,'caller')                    %plot probe background graph
    Datac(:,:,iset)=int8(ceil(cdata)-128);      %setup data for file
elseif setting==9                               %Scan Aborted, save incomplete scan.
    disp(['incomplete scan ' num2str(scan) ' saved as:' textdisplay])
    save(textdisplay,'Dataa','Datab','Datac')   %SAVE SCAN DATA.
end
intent=['Intensity = ' num2str(inten,3) '    '];%Calculate intensity
set(handles.textdisplay,'String',[intent textdisplay])%Display the text
handles.value.picwidth=w;                       %Save new picwidth
set(handles.picwidth,'String',handles.value.picwidth*2); guidata(handles.picwidth,handles)
drawnow

function RectFn(x,y)
picdisp(zeros(480,640),guidata(gcf),-1,'Crosscorrelation Fit',0)

function camerasat(handles,file,scan,iset)
set(handles.csat,'Visible','on')
delays=handles.value.delays;
fid=fopen([file(1:end-6) '.txt'],'a');          %WRITE TO A FILE
csattxt=['Camara Saturated on scan no. ' num2str(scan) ' and delay time' num2str(delays(iset-1))];
errordlg(csattxt)
fprintf(fid,csattxt);
fclose(fid)
handles.value.csatfile=[file(1:end-6) '.txt'];

%% 'Load Settings' Function
function Load_Settings(hObject,handles,FileName,PathName)
global spec Running ExitScan saveno numav livethresh imthresh symon maskc
load name; load dataf; load piccentre
load([PathName FileName])
spec=0;Running=0;ExitScan=0;saveno=0;numav=3;livethresh=0;
symon=get(handles.SymImage,'Value');
maskc=get(handles.maskc,'Value');
prevcontrast_Callback(hObject, 0, handles)%get image contrast setting
%Set the settings values to be the same as those in the file
    handles.value.scanbackwards = scanbackwards;
    handles.value.scannum       = scannum;
    handles.value.imagetime     = imagetime;
    handles.value.framerate     = framerate;
    handles.value.camcontrast   = camcontrast;
    handles.value.stagestep     = stagestep;
    handles.value.stageend      = stageend;
    handles.value.exppoints     = exppoints;
    handles.value.expstageend   = expstageend;
    handles.value.stagestart    = stagestart;
    handles.value.btime         = btime;
    handles.value.ctime         = ctime;
    handles.value.backsub       = backsub;
```

```
    handles.value.delays        = delays;
    handles.value.StepCC        = StepCC;
    handles.value.home          = hom;

%These are not saved in the file but need to be set anyway
    imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'
    handles.value.csatfile      = '';
    handles.value.shutters      = 0;
    handles.value.picwidth      = wd;
    handles.value.piccentre     = cn;
    handles.value.c             = [];

%Tell the gui window what to display
    set(handles.scanbackwards,'Value' ,handles.value.scanbackwards);
    set(handles.scannum,      'String',handles.value.scannum);
    set(handles.imagetime,    'String',handles.value.imagetime);
    set(handles.stagestep,    'String',handles.value.stagestep);
    set(handles.stageend,     'String',handles.value.stageend);
    set(handles.exppoints,    'String',handles.value.exppoints);
    set(handles.expstageend,  'String',handles.value.expstageend);
    set(handles.stagestart,   'String',handles.value.stagestart);
    set(handles.btime,        'String',handles.value.btime);
    set(handles.ctime,        'String',handles.value.ctime);
    set(handles.backsub,      'Value' ,handles.value.backsub);
    set(handles.StepCC,       'String',handles.value.StepCC);
%These are not saved in the file but need to be displayed anyway
    set(handles.piccentretxt, 'String',[num2str(cn(1)) ', ' num2str(cn(2))])
    set(handles.runcounter,   'String',Count);
    set(handles.displaydate,  'String',['Date: ' date]);
    set(handles.textdisplay,  'String','')%makes the text display blank
    set(handles.picwidth,     'String',handles.value.picwidth*2);
    set(handles.stageslider,  'Min'   ,handles.value.home/0.0002998)
    set(handles.stageslider,  'Max'   ,handles.value.home+500000)
    set(handles.stageslider,  'Value' ,0)
    set(handles.numav,        'String',numav)
    set(handles.imthresh,     'Value', imthresh/10)
    set(handles.txtthresh,    'String',num2str(imthresh,2))
% Save Values and Update handles structure
    guidata(handles.textdisplay, handles)
    save('dataf','data','wd')
%% '
%% USER INTERFACE
%% Editbox Function
%gets text from an editbox, checks it's good, and returns the inputted value
%if it's good, or the previous value if it's not
%hObject, handles to the editbox
%hValue, previous good value entered by user
%min, minimum acceptable answer
%max, maximum acceptable answer
%int, TF whether input must be an intiger
function value=edit(hObject,hValue,min,max,int,string)
value = str2double(get(hObject, 'String'));%get the new scannum value
if int, r=rem(value,1)>0;i='an intiger';else r=0;i='a number';end %if value needs to be an intiger
if isnan(value)||value<min||value>max||r
    value = hValue; set(hObject, 'String', value);
    errordlg([string ' Input must be ' i ' between ' num2str(min) ' and ' num2str(max)],'Error');
end

%% picdisp1
function picdisp1(rdata,handles,textdisplay)
global wd cn crson mw mask mi symon maskc
%rdata      = raw data to be made into cropped picture and graph
%textdisplay= string to display in window
%handles    = all the variables and values
%get and calculate variables
load piccentre
cn   = handles.value.piccentre;
wd   = round(min([cn(1) cn(2) 640-cn(1) 480-cn(2) wd]));%width is the shortest distance from the centre to the edge
```

```
[x,y]= ndgrid((1:480)-cn(2),(1:640)-cn(1)); mask=(x.^2+y.^2)<mw^2;%calculate mask
set(handles.piccentretxt, 'String', [num2str(cn(1)) ', ' num2str(cn(2))])
crson= get(handles.piccross,'Value');

if ~get(handles.campreview,'Value') %if not previewing
    if maskc, rdata(mask) = rdata(mask)*mi; end         %mask picture
    if symon, cdata = symetrise(rdata,cn(2),cn(1),wd);  %symatrise picture
    else cdata= rdata(cn(2)-wd+1:cn(2)+wd-1,cn(1)-wd+1:cn(1)+wd-1); end %crop picture
    lw   = min([floor(wd/2) 8]);                        %line width
    up   = mean(cdata(wd:-1:1, wd-lw:wd+lw),2);         %blue(mean(...,2)=avg the rows)
    down = mean(cdata(wd:(2*wd-1),wd-lw:wd+lw),2);      %green
    left = mean(cdata(wd-lw:wd+lw,wd:-1:1),1);          %red(mean(...,1)=avg the columns)
    right= mean(cdata(wd-lw:wd+lw,wd:(2*wd-1)),1);      %turquose
    inten= mean(mean(cdata(:,:)));
    if crson,cdata(:,wd-1:wd+1)=NaN; cdata(wd-1:wd+1,:)=NaN;end %make centre cross
    axes(handles.Graph)
    plot(1:wd,up,1:wd,down,1:wd,left,1:wd,right); xlim([0 wd])%plot graph
    xlabel('distance from centre (pixels)'); ylabel('Intensity'); title('Energy Distribution')
    legend('up','down','left','right','Location','NorthOutside')%create key
    textdisplay=[textdisplay 'Centre: (' num2str(cn(1)) ', ' num2str(cn(2)) ')']; %display centrepoint
    axes(handles.Picture);pcolor(cdata);colormap jet  %Display the picture
    shading interp; axis image off; view(0,90)
    inten=['Intensity = ' num2str(inten,3) '%    '];%Calculate intensity
    set(handles.textdisplay,'String',[inten textdisplay])%Display the text
    handles.value.picwidth=wd;                      %Save new picwidth
    set(handles.picwidth,'String',handles.value.picwidth*2); guidata(handles.picwidth,handles)
    set(handles.maskinten,'String',mi);
    set(handles.maskwidth,'String',mw);
    guidata(handles.picwidth,handles)
end

function symdata=symetrise(data,c1,c2,w)
symdata=data(c1-w+1:c1+w,c2-w+1:c2+w);%crop data
rDD=symdata(w+1:2*w,w+1:2*w,:)+symdata(w:-1:1,w:-1:1,:)+symdata(w+1:2*w,w:-1:1,:)+symdata(w:-1:1,w+1:2*w,:);
symdata(1:w,1:w,:)=rDD(w:-1:1,w:-1:1,:);
symdata(1:w,2*w:-1:w+1,:)=symdata(1:w,1:w,:);
symdata(2*w:-1:w+1,:,:)=symdata(1:w,:,:);
%% PICTURE SECTIONBOX
% 'Centre' Button
function piccentre_Callback(hObject, eventdata, handles)
piccentre(hObject, eventdata, handles)
function piccentre(hObject, eventdata, handles)
load piccentre,load dataf
if get(handles.campreview,'Value'),axis=handles.Graph; W=[640 480]; else axis=handles.Picture; W=[wd*2 wd*2];end
set(handles.textdisplay,'String','Please click within picture')
r=waitforbuttonpress;               %wait for user to click mouse
while r==1,r=waitforbuttonpress;end %if user presses a keyboard button keep on waiting
cp= get(axis,'CurrentPoint');       %get where they clicked their mouse
x = round(cp(1,1));             y = round(cp(1,2));
a = handles.value.piccentre(1); b = handles.value.piccentre(2);

for i=1:3 %if user clicked outside the picture, send them back to click again
    if x<=0 || x>=W(2) || y<=0 || y>=W(1)
        errordlg('please click within picture','Center');uiwait,
        r=waitforbuttonpress;
        while r==1,r=waitforbuttonpress;end %if user presses a keyboard button keep on waiting
        cp= get(axis,'CurrentPoint');       %get where they clicked their mouse
        x = round(cp(1,1)); y = round(cp(1,2));
    end
end
a=round(a-wd+x); b=round(b-wd+y);         %calculate absolute centre using old width
wd= min([a b 640-a 480-b wd]);              %calculate new width
cn=[a b]; handles.value.piccentre=cn;
handles.value.picwidth = wd;
guidata(hObject,handles)                %save point
save('piccentre','wd','cn','mw','mi')
picdisp1(data,handles,'')               %go to 'Picture Function'
```

```
%Center Lines Tickbox
function piccross_Callback(hObject, eventdata, handles)
load dataf, picdisp1(data,handles,'')

%Mask Area Tickbox
function maskc_Callback(hObject, eventdata, handles)
global maskc, maskc=get(hObject,'Value');
load dataf, picdisp1(data,handles,'')

%Centroid Tickbox
function CentroidIm_Callback(hObject, eventdata, handles)
global centon, centon=get(hObject,'Value');
load dataf, picdisp1(data,handles,'')

function SymImage_Callback(hObject, eventdata, handles)
global symon, symon=get(hObject,'Value');
load dataf, picdisp1(data,handles,'')

function maskwidth_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function maskwidth_Callback(hObject, eventdata, handles)
global mw, load dataf, load piccentre
value=str2double(get(hObject,'String'));
if isnan(value)||value<0, errordlg('value must be a positive number'),set(hObject,'String',mw),return,end
mw=value; save('piccentre','wd','cn','mw','mi')
picdisp1(data,handles,'')

function maskinten_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function maskinten_Callback(hObject, eventdata, handles)
global mi, load dataf, load piccentre
value=str2double(get(hObject,'String'));
if isnan(value)||value<0, errordlg('value must be a positive number'),set(hObject,'String',mi),return,end
mi=value; save('piccentre','wd','cn','mw','mi')
picdisp1(data,handles,'')

% Picture Arrow Buttons
function pcentreUP_Callback (hObject,eventdata,handles),arrow(hObject,handles,2,-1)
function pcentreDOWN_Callback(hObject,eventdata,handles),arrow(hObject,handles,2,+1)
function pcentreLEFT_Callback(hObject,eventdata,handles),arrow(hObject,handles,1,-1)
function pcentreRIGHT_Callback(hObject,eventdata,handles),arrow(hObject,handles,1,+1)
function arrow(hObject,handles,ci,pn)%ci=1 OR 2, pn=+1 OR -1
set(hObject,'Enable','off')
load piccentre,load dataf%cn=handles.value.piccentre;
cn(ci)=cn(ci)-1*pn;         %change centre point by 1 pixel
w = handles.value.picwidth;                              %old width
wd= min([cn(1) cn(2) 640-cn(1) 480-cn(2) w]);            %calculate new width
handles.value.picwidth = wd;
handles.value.piccentre= cn; guidata(hObject,handles)    %save value
save('piccentre','wd','cn','mw','mi')
picdisp1(data,handles,'')                             %go to 'Picture Function'
set(hObject,'Enable','on')

% Width
function                    picwidth_CreateFcn(hObject,                    eventdata,                    handles),if
ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','
white');end %set background colour
function picwidth_Callback(hObject, eventdata, handles)
load dataf,load piccentre
%get data
picwidth = str2double(get(hObject, 'String'));           %get the 'Width' value
wd=floor(picwidth/2);                                    %work with 1/2 the width (rounded down)
x=handles.value.piccentre(1);                            %piccenter values (x,y)
y=handles.value.piccentre(2);
w=handles.value.picwidth;                                %previous width (in case user enters faulty width)
W=[480 640];                                             %width of original data
[m I] = min([x y W(2)-x W(1)-y]);                        %shortest distance from the centre point to the edge
```

```
if isempty(get(hObject, 'String'))
    handles.settings.picwidth=m;
%error messages:
elseif isnan(picwidth)                                  %if contains text or charicters
    set(hObject, 'String', w*2)                         %go back to previous value
    errordlg('Input must be a number','Error'),uiwait   %display error message
elseif any(picwidth > W)                                %if it's too big
    set(hObject, 'String', w*2)                         %go back to previous value
    errordlg(['Input cannot be above maximum picture width (' num2str(min(W)) ')'],'Error'),uiwait%display error
message
elseif picwidth < 10                                    %if it's too small
    set(hObject, 'String', w*2)                         %go back to previous value
    errordlg('Picture width must be greater than 9','Error'),uiwait%display error message
%save data:
elseif wd > m                                          %if image too large for centre point to work
    button=questdlg('Making the picture this large will move the centre point. Do you want to continue?');
    if strcmp(button,'No'),set(hObject, 'String', w*2)  %if user presses 'No', go back to previous value
    else                                                %if user presses 'Yes', calculate new piccentre:
        for i=1:2                                       %for both x and y
            switch I                                    %change smallest centrepoint to accommidate larger width
                case 1, x=wd;
                case 2, y=wd;
                case 3, x=W(2)-wd;
                case 4, y=W(1)-wd;
            end
            [m I] = min([x y W(2)-x W(1)-y]);           %find next smallest centrepoint
            if wd <= m, break, end                      %if it's ok, carry on. Otherwise change it too.
        end
        cn=[x y]; handles.value.piccentre= cn;          %Save the new piccentre
        handles.value.picwidth = wd;                    %Save the new picwidth
    end
else                                                    %If there are no problems
    handles.value.picwidth = wd;                        %Save the new picwidth
end
save('dataf','data','wd') %Save the new picwidth value
save('piccentre','wd','cn','mw','mi')
guidata(hObject,handles)
picdisp1(data,handles,'')      %go to 'Picture Function'


%% IMAGE SETTINGS Sectionbox
% Time to Aquire an image
function imagetime_CreateFcn(hObject, eventdata, handles)
if ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end %set background colour
function imagetime_Callback(hObject, eventdata, handles)
imagetime = str2double(get(hObject, 'String'));%get the new imagetime value
if isnan(imagetime)
    set(hObject, 'String', handles.value.imagetime);
    imagetime = handles.value.imagetime; errordlg('Input must be a number','Error');
end
handles.value.imagetime = imagetime; guidata(hObject,handles)% Save the new imagetime value

% handles.value.imagetime=edit(hObject,handles.value.imagetime,0,Inf,0,'Image Time');
% guidata(hObject,handles)

% Background Subtract (checkbox)
function backsub_Callback(hObject, eventdata, handles)
backsub = get(hObject, 'Value');%get the new backsub value
if backsub, set([handles.btime handles.ctime handles.backtxt], 'Enable','off')
else        set([handles.btime handles.ctime handles.backtxt], 'Enable','on' ),end
% Save the new backsub value
handles.value.backsub = backsub; guidata(hObject,handles)
imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'

% Background Aquire Time
```

```
function ctime_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function ctime_Callback(hObject, eventdata, handles)
backtime = str2double(get(hObject, 'String'));%get the new backtime value
if isnan(backtime)
    set(hObject, 'String', handles.value.ctime);
    backtime = handles.value.ctime;
    errordlg('Input must be a number','Error');
end
handles.value.ctime = backtime; guidata(hObject,handles)% Save the new btime value

% Background Aquire Time
function btime_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white'),end
function btime_Callback(hObject, eventdata, handles)
backtime = str2double(get(hObject, 'String'));%get the new backtime value
if isnan(backtime)
    set(hObject, 'String', handles.value.btime);
    backtime = handles.value.btime;
    errordlg('Input must be a number','Error');
end
handles.value.btime = backtime; guidata(hObject,handles)% Save the new btime value

% handles.value.btime=edit(hObject,handles.value.btime,0,Inf,0,'Pump Alone Time');
% guidata(hObject,handles)

% Camera Saturating Button (Red button that only appears if camera is saturating)
function csat_Callback(hObject, eventdata, handles)
set(hObject,'Visible','off')
if ~isempty(handles.value.csatfile),open(handles.value.csatfile),end


function Graph_ButtonDownFcn(hObject, eventdata, handles)


% Crosscorelation fit listbox
function CCfit_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function CCfit_Callback(hObject, eventdata, handles)
global spec, if size(spec)~=5, picdisp(zeros(480,640),handles,-1,'Crosscorrelation Fit',0),end

%% crosscorrelation fitting function
function yfit=tracefit(x,y,handles)
persistent d
%inputs
yfit = [];    dd     = find(y);                    %get non-zero values of y
yy   = y(dd); y(y==0)= yy(end);                   %predict data if there are zero points
txt  = '';    a      = sort(yy);
x    = x(:);  yofs   = mean(a(1:3));               %get y offset
y    = y(:);  ymax   = mean(a(end-3:end));
dsign=1-2*get(handles.scanbackwards,'Value');
fits = get(handles.CCfit,'Value');  %make sure x and y are always column vectors
if length(y(y~=0))<5, fits=1; txt='Not enough Data points'; end %don't fit till there is enough data points
%DO FITTING
%no crosscorrelation fitting
if fits==1; axes(handles.Delays), cla(gca)          %clear last fit
%gaussian crosscorrelation
elseif fits==2;
    %fit data
    c    = fit(x(dd),(y(dd)-yofs),fittype('gauss1'));%calculate fit ('gauss1' uses Y = a1*exp(-((x-b1)/c1)^2) to fit
data)
    yfit= c.a1*exp(-((x-c.b1)/c.c1).^2)+yofs;       %maxe yvalues for plotting fit
    er  = confint(c,0.99);                          %get errors (to 99% confidence)
    cct = c.c1*2.35482/sqrt(2);                     %get FWHM (sqrt(2) gets std.dev. 2.35482 turns st.dev. into FWHM))
    cer = (er(2,2)-er(1,2))/2;                      %extract +- error for centre
    wer = (er(2,3)-er(1,3))*0.8325546;              %extract +- error for FWHM
    txt = ['CC = ' num2str(cct,'%1.1f') '+-' num2str(wer,'%1.1f') ' fs.\nStage Centre = ' num2str(dsign*c.b1,'%1.1f')
'+-' num2str(cer/sqrt(2),'%1.1f') ' fs.'];
```

```
%rise time crosscorrelation
elseif fits==3;
    flip = 0;if yy(end-1)<yy(1), y = flipud(y); x = flipud(-1*x); flip=1;end %if drop rather than rise
    [m i]= max(y); y(i+1:end) = ymax;                  %only fit the rising edge (i.e. if it rises then dacays again,
this gets rid of the decay)
    d    = [0 0.2 yofs (max(y)-yofs)/1000];
    opt  = optimset('TolX',1e-20000,'TolFun',1e-1200,'MaxFunEvals',700);
    ccfit= @(d,x) 500*d(4)*(1+erf((x/1000-d(1))/(sqrt(2)*d(2)/2.35482)))+d(3); %Cumulative Gaussian Anonymous Fn (note
2.35482 is the conversion factor between the gaussian 'varience' and a FWHM)
    lb   = [-Inf 0 -Inf 0]; ub = [Inf 2 Inf Inf];   %set lower and upper bounds
    [d err] = lsqcurvefit(ccfit,d,x,y,lb,ub,opt);   %fit using Cumulative Gaussian Anonymous Fn
    yfit = ccfit(d,x);                              %get red fit line
    cc   = d(2)*1000; cn = d(1)*1000; err = err/mean(y)^2;%get values and percentage error
    if flip, yfit = flipud(yfit); cn = -cn; end     %if drop rather than rise, flip the output fit line back
    txt  = ['CC = ' num2str(cc,3) ' fs.\nStage Centre = ' num2str(dsign*cn,3) ' fs.'];
    %txt    =  ['CC  =  '  num2str(cc,3)  '+-'  num2str(cc*err,'%1.1f')  '  fs.\nCentre  =  '  num2str(cn,3)  '+-'
num2str(cn*err,'%1.1f') ' fs.'];
end
set(handles.fittxt,'String',sprintf(txt))           %display the fitting text

%% STAGE SETTINGS Sectionbox

% Scan Backwards.
function scanbackwards_Callback(hObject, eventdata, handles)

% Number of Scans Taken
function                    scannum_CreateFcn(hObject,                    eventdata,                    handles),if
ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','
white');end %set background colour
function scannum_Callback(hObject, eventdata, handles)
handles.value.scannum = edit(hObject,handles.value.scannum,1,Inf,1,'Number of Scans');  %go to Edit Box fn
guidata(hObject,handles)

% Stage Step
function                    stagestep_CreateFcn(hObject,                    eventdata,                    handles),if
ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','
white');end %set background colour
function stagestep_Callback(hObject, eventdata, handles)
handles.value.stagestep = edit(hObject,handles.value.stagestep,1,Inf,0,'Step Size');  %go to Edit Box fn
guidata(hObject,handles)
imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'

% Stage Starting Point
function                    stagestart_CreateFcn(hObject,                    eventdata,                    handles),if
ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','
white');end %set background colour
function stagestart_Callback(hObject, eventdata, handles)
handles.value.stagestart                                                                                    =
edit(hObject,handles.value.stagestart,ceil(handles.value.home/0.0002998),handles.value.stageend,0,'Starting   Point');
%go to Edit Box fn
guidata(hObject,handles)
imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'

% Linear End Point
function stageend_CreateFcn(hObject, eventdata, handles)
if
ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','
white');end %set background colour
function stageend_Callback(hObject, eventdata, handles)
if             handles.value.exppoints             ~=0,             max=handles.value.expstageend;             else
max=floor(get(handles.stageslider,'Max')/0.0002998); end
handles.value.stageend  =  edit(hObject,handles.value.stageend,handles.value.stagestart,max,0,'End  of  Scan');   %go  to
Edit Box fn
guidata(hObject,handles)
imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'

% Exponential Points
```

```
function exppoints_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function exppoints_Callback(hObject, eventdata, handles)
handles.value.exppoints = edit(hObject,handles.value.exppoints,0,Inf,1,'Number of Exponential Steps'); %go to Edit
Box fn
if handles.value.exppoints==0, set(handles.expstageend,'Enable','off')
else set(handles.expstageend,'Enable','on'),end
guidata(hObject,handles)
imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'

% Exponential End Point
function expstageend_CreateFcn(hObject, eventdata, handles),if
ispc&&isequal(get(hObject,'BackgroundColor'),get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','
white');end %set background colour
function expstageend_Callback(hObject, eventdata, handles)
handles.value.expstageend=edit(hObject,handles.value.expstageend,handles.value.stageend,floor(get(handles.stageslider,
'Max')/0.0002998),0,'End of Scan'); %go to Edit Box fn
guidata(hObject,handles)
imagenum_Calculate(hObject, handles); %go to 'Calculate Number of Images'

%% HARDWARE Sectionbox
% Shutter & Theshold Stuff
% Pump Shutter Button
function shutter1_Callback(hObject, eventdata, handles)
err='';
if ~handles.value.shutters,h=helpdlg('Please Wait, Setting up shutters');err=setupshutters(handles);close(h),end
%initialise shutters if required
if isempty(err)
    if get(hObject,'Value');shutterop(handles,1,1) %Open shutter
    else shutterop(handles,1,0),end %Close shutter
else helpdlg(err)
end

% Probe Shutter Button
function shutter2_Callback(hObject, eventdata, handles)
err='';
if ~handles.value.shutters,h=helpdlg('Please Wait, Setting up shutters');err=setupshutters(handles);try
close(h),end,end %initialise shutters if required
if isempty(err)
    if get(hObject,'Value');shutterop(handles,2,1) %Open shutter
    else shutterop(handles,2,0),end %Close shutter
else helpdlg(err)
end

% Threshold Slider
function imthresh_CreateFcn(hObject, eventdata, handles),if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function imthresh_Callback(hObject, eventdata, handles)
global imthresh
load Settings/Default_Settings.mat
imthresh=10*get(handles.imthresh,'Value');
set(handles.txtthresh,'String',num2str(imthresh,2))
clear hObject eventdata handles, save Settings/Default_Settings.mat

% Stage Slider and Buttons
%Slider
function stageslider_CreateFcn(hObject, eventdata, handles),if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor',[.9 .9 .9]);end
function stageslider_Callback(hObject, eventdata, handles)
global stage stageaxis
value=get(hObject,'Value');
err=setupstage(handles,1);%go to setupstage function
if err==0;MOV(stage,stageaxis,value);stagepos(handles),end

% Set New Home
function zerostage_Callback(hObject, eventdata, handles)
global stage stageaxis
load Default_Settings
```

```
h=questdlg('Set the current stage position as the new Zero point?');
if strcmp(h,'Yes')
    err=setupstage(handles,1);%go to setupstage function
    DFH(stage,stageaxis); %Define Home
    handles.value.home=qTMN(stage,stageaxis);%Get new home value
    guidata(hObject,handles)
    stagepos(handles)
    hom=handles.value.home;
    %save the new stage home value in default settings
    save('Default_Settings','imthresh','scanbackwards','scannum','imagetime','framerate',...
        'camcontrast','stagestep','stagestart','stageend','exppoints',...
        'expstageend','backtime','backsub','delays','hom','StepCC')
    helpdlg('Stage Position Saved','')
end

% Step Size Editbox
function StepCC_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function StepCC_Callback(hObject, eventdata, handles)
StepCC=str2double(get(hObject,'String'));%get the new StepCC value
step=0.0002998*(StepCC/500000);
if isnan(StepCC)||StepCC<=0||step>0.1
    set(hObject, 'String', handles.value.StepCC);
    StepCC = handles.value.StepCC;
    errordlg('Input must be a positive number not larger than 1/10th of the stage','Error');
end
% Save the new StepCC value
handles.value.StepCC = StepCC; guidata(hObject,handles)

% handles.value.StepCC=edit(hObject,handles.value.StepCC,0,500000000,0,'Stage Step Size');
% guidata(hObject,handles)

stagepos(handles)

%Go To Editbox
function GoToCC_CreateFcn(hObject, eventdata, handles),if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function GoToCC_Callback(hObject, eventdata, handles)
global stage stageaxis
err=setupstage(handles,1);%go to setupstage function
if err==0;
    GoToCC=str2double(get(hObject,'String'))*0.0002998;%get the value in mm
    min=round(qTMN(stage,stageaxis));               %get min and max stage values
    max=round(qTMX(stage,stageaxis));
    if isnan(GoToCC)||GoToCC<min||GoToCC>max,errordlg('Input must be a number not larger than the ends of the
stage','Error');end
    MOV(stage,stageaxis,GoToCC); stagepos(handles)  %move stage
end

% Current Stage Position Functions
function stagepos(handles)                          %get stage position function
global stage stageaxis                              %set up variables
err=setupstage(handles,1);%go to setupstage function
if err==0
    set(handles.textdisplay,'String','stage moving...')
    %set up slider
    min=round(qTMN(stage,stageaxis));                   %get min and max stage values
    max=round(qTMX(stage,stageaxis));
    step=handles.value.StepCC*0.0002998/(max-min);      %get slider step size
    set(handles.stageslider,'Min',min)                  %set min and max stage values
    set(handles.stageslider,'Max',max)
    set(handles.stageslider,'SliderStep',[step step*10])    %set slider step size
    stagemove(stage,stageaxis,handles,1)                %display dynamic position while stage is moving
    %display position and limits once stage has stopped
    a=qPOS(stage,stageaxis);                            %get final position in mm
    b=TranslateError(stage,qERR(stage));                %get any errors
    set(handles.textdisplay,'String',['Final Position: ' num2str(a/0.0002998,'%11.0f') ...
        ' fs. Stage can be moved between: ' num2str(min/0.0002998,'%11.0f')...
```

```
        ' and ' num2str(max/0.0002998,'%11.0f') '. ' b])      %display stage limits
end

%function to delay program until stage has finished moving and update position as it moves
function stagemove(stage,stageaxis,handles,slider)
while stage.ismoving(stageaxis)
    dispstpos(stage,stageaxis,handles,slider)           %display position as moving
    pause(0.05)                                         %wait a bit
end
dispstpos(stage,stageaxis,handles,slider)               %display final position

%function to display position of stage
function dispstpos(stage,stageaxis,handles,slider)
b=qPOS(stage,stageaxis);                                %get position
set(handles.stagepostxt, 'String',num2str(b/0.0002998,'%11.0f')),%display position
if slider, set(handles.stageslider,'Value',b),end       %move slider
%% '
%% MENUS
%% File Menu
function FileMenu_Callback(hObject, eventdata, handles)
function SaveSettings_Callback(hObject, eventdata, handles),savesettings(handles)%savecentre(handles),
function EditColormap_Callback(hObject, eventdata, handles),colormapeditor
function QQuit_Callback(hObject, eventdata, handles),global cl,cl=4;close

function EQuit_Callback(hObject, eventdata, handles)%Emergancy Quit
%try to shut down hardware
global stage vid;   s='';
try flushdata(vid),delete(vid),s='Camera Shut Succesfully, ';  catch s='Camera NOT Shut Succesfully, '; end
try CloseConnection(stage),s=[s 'Stage Shut Succesfully, ']; catch s=[s 'Stage NOT Shut Succesfully, ']; end
try handles.activex2.StopCtrl; handles.activex3.StopCtrl; s=[s 'Shutters Shut Succesfully'];catch s=[s 'Shutters NOT
Shut Succesfully'];end %Shutters
%close software
delete(gcf),errordlg([s '. Camera_Stage closed down abruptly. You may need to restart MATLAB to use it again'])
clear all

function LoadSettings_Callback(hObject, eventdata, handles)
C={'name.mat' 'piccentre.mat' 'dataf.mat'};
[FileName, PathName, FilterIndex]=uigetfile('*.mat','Pick a new Settings file','Settings/Default_Settings.mat');
if FilterIndex==0
    msgbox('You have not chosen a file. Settings will stay the same','Load Settings','error')
elseif max(strcmp(FileName,C))
    msgbox('Invalid Settings File. Settings will stay the same','Load Settings','error')
else
    Load_Settings(hObject,handles,FileName,PathName)%go to 'Load Function'
    msgbox('Settings have been Changed','Load Settings','help')
end

% Ask if user wants to save settings
function savesettings(handles)
global imthresh
q=questdlg(sprintf('Do you want to save your settings?\n\nSaving the settings saves all the camera and stage settings,
and the stage zero point, to a file for you to load again another time. If you save over Default_Settings.mat the
settings you have entered will become the new default settings you get every time you open the program. If you don''t
want this, give it a different file name.'),'Save','Yes');
if strcmp(q,'Yes')

    [FileName, PathName, FilterIndex]=uiputfile('Settings/Default_Settings.mat','Select file to write');
    if FilterIndex==0
        msgbox('You have not chosen a file. Settings will not be saved','Save Settings','error')
    else
        %Read the settings from the UI
        scanbackwards   = get(handles.scanbackwards, 'Value');
        scannum         = str2double(get(handles.scannum, 'String'));
        imagetime       = str2double(get(handles.imagetime, 'String'));
        framerate       = handles.value.framerate;
        camcontrast     = handles.value.camcontrast;
        stagestep       = str2double(get(handles.stagestep, 'String'));
        stagestart      = str2double(get(handles.stagestart, 'String'));
```

```
        stageend        = str2double(get(handles.stageend, 'String'));
        exppoints       = str2double(get(handles.exppoints, 'String'));
        expstageend     = str2double(get(handles.expstageend, 'String'));
        btime           = str2double(get(handles.btime, 'String'));
        ctime           = str2double(get(handles.ctime, 'String'));
        backsub         = get(handles.backsub, 'Value');
        StepCC          = str2double(get(handles.StepCC, 'String'));
        delays          = handles.value.delays;
        hom             = handles.value.home;
        imthresh        = imthresh;
        %Write the settings to the file
        save([PathName FileName],'scanbackwards','scannum','imagetime','imthresh',...
            'framerate','camcontrast','stagestep','stagestart','stageend',...
            'exppoints','expstageend','btime','ctime','backsub','delays','hom','StepCC')
        h=msgbox(['Settings have been Saved as: "' FileName '"'],'Save Settings','help');
        uiwait(h)
    end
elseif strcmp(q,'Cancel')
    return
end


%% Hardware Menu
function HardwareMenu_Callback(hObject, eventdata, handles)
function TestHardware_Callback(hObject, eventdata, handles),Test_Hardware
function cammcontrast_Callback(hObject, eventdata, handles)%change the camera contrast
cn = inputdlg('Choose Camera Contrast (180-1020)','',1,{num2str(handles.value.camcontrast)});
if isempty(cn) %see if they chose a sensible value, then save it.
else cn = str2num(cell2mat(cn));
    if isnan(cn)||cn<180||cn>1020,errordlg('Please choose a positive intiger between 180 and 1020')
    else handles.value.camcontrast=cn;guidata(hObject,handles),helpp('Camera Settings Changed'),end
    clear cn hObject eventdata handles, save Default_Settings.mat
end

function cammframerate_Callback(hObject, eventdata, handles)
load Default_Settings.mat
[fr       ok]       =       listdlg('ListString',{'60','30','15','7.5','3.7'},'SelectionMode','single','ListSize',[160
70],'PromptString',{'Choose the Camera Framerate','(Frames per second)'},'InitialValue',handles.value.framerate);
if ok, handles.value.framerate=fr; guidata(hObject,handles),helpp('Camera Settings Changed'),end
clear fr ok hObject eventdata handles, save Default_Settings.mat

%% Help Menu
function HelpMenu_Callback(hObject, eventdata, handles)
function GeneralHelp_Callback(hObject, eventdata, handles),          helpp('Help!')
function TimingHelp_Callback(hObject, eventdata, handles),           helpp('Hardware')
function PreviewHelp_Callback(hObject, eventdata, handles),          helpp('Preview')
function Stage_SettingHelp_Callback(hObject, eventdata, handles),    helpp('Stage Settings')
function Camera_SettingHelp_Callback(hObject, eventdata, handles),   helpp('Camera Settings')
function PictureHelp_Callback(hObject, eventdata, handles),          helpp('Picture')
function Naming_ConventionHelp_Callback(hObject, eventdata, handles),helpp('Naming Convention')
function AboutHelp_Callback(hObject, eventdata, handles),            helpp('About')
function ProgrammingHelp_Callback(hObject, eventdata, handles),      helpp('Programming')

%% Help Files
function helpp(title)
file='Camera_Stage';
if strcmp(title,'Help!')
    a=help(file);%get the first few commented lines from Camera_Stage.m
    b=isspace(a);a(b)=' ';%get rid of artificial newlines so text wraps nicely
    c=strfind(a,a(1:5));%get rid of first few lines of nonsense(if they exist)
    c(end+1:end+3)=1;%if no lines on nonsense, start from beginning
    helptxt=sprintf(a(c(3):end));

elseif strcmp(title,'Hardware');
helptxt=sprintf(['Hardware Sectionbox:\n\n'...
    'This section is for you to overlap the pump and probe beams temporally and '...
    'spacially by moving the stage and operating the shutters. You can move the stage '...
    'manually by setting a step size and hitting the arrow buttons. '...
```

```
    '(If you run "Preview" you can see the signal while you do this) '...
    'When you think you are in the right area you can scan the stage using '...
    '"Run". If "Gaussian" in pressed, the width and centre point of the '...
    'fit will appear below the image. Once the Run is finished, use the arrow '...
    'buttons to get the stage to the centre postion then use "New Home" to '...
    'set the new zero postion for the stage.\n\nStep Size : determines the '...
    'size of the step.\n  <  : moves the stage backwards 1 step size.\n  >  : '...
    'moves the stage forward one step size.\n(if you click in the trough you '...
    'move 10x the step size)\nGo Home: moves the stage to it''s zero position.'...
    '\nNew Home : set a new zero point for the stage.\nPosition: displays the current '...
    'position of the stage. \n\nShutter Open/Close: Controls the two shutters. '...
    '\n(When the button is depressed, the shutter is open)\nImage Threshold: controls '...
    'how much of the camera noise you cut out by deleting any signal below this threshold. '...
    'Be aware that if you set it too high you may start to cut out real data as '...
    'well as noise.\n\nWhen you press any of these buttons for the first time after '...
    'opening the program, the program will attempt to set up the hardware. '...
    'Please be patient!\n\n If you are having problems with the hardware, try '...
    'Closing the program, then unplugging and replugging the USB cord on the '...
    'offending item. ''Test Hardware'' in the File Menu can be useful']);

elseif strcmp(title,'Stage Settings')
helptxt=sprintf(['Stage Settings Sectionbox:\n\n'...
    'These settings determines how the stage moves to change the time delay'...
    ' between the pump and the probe. The time delay can be changed in regular'...
    ' steps (Linear step size) up until a certain point (Linear end point), and'...
    ' then in exponentially increasing steps until the Exp end point. This'...
    ' allowes you to get more points where the interesting things are happening'...
    ', but still scan to long time delays without taking forever. If you don''t'...
    ' want to use exponential steps, then set the No. of Exp point to zero.'...
    ' The Fit Gaussian Button fits a gaussian to the delays. The number of'...
    ' steps per scan tells you how many steps you have.'...
    ' Remember to times this by the number of scans you will take to'...
    ' get the total number of steps. The Scan Backwards Tickbox allows you to'...
    ' scan in a negative direction. This is usefull if is your pump this is being'...
    ' delayed rather than your probe']);


elseif strcmp(title,'Camera Settings')
helptxt=sprintf(['Camera Setting Sectionbox:\n\n'...
    'These settings determines how the camera functions and takes images.'...
    ' \nMain Aquire Time (s): determines how long the program collects data'...
    ' from the camera and adds it together to make an integrated picture.'...
    ' \nBackground Subtract off will, if ticked, stop the program from'...
    ' collecting pump alone and probe alone images in addition to the main'...
    ' images. This may be useful if you just want to take a quick scan to'...
    ' get the cross correlation. \nBackground Time (s): is the same as Main'...
    ' Aquire Time but for the pump alone and probe alone images. \nFrame'...
    ' Rate (/s): is one of the settings of the camera that can be controlled.'...
    ' Please choose a value fromthe drop down list. \nContrast: is another'...
    ' camera setting. If your image is too dark or is saturating, this'...
    ' setting may be useful.']);

elseif strcmp(title,'Picture')
helptxt=sprintf(['Picture Settings:\n\n'...
    'This section helps you set a centrepoint for the image. If you set'...
    ' a good centrepoint before a Run, then you will have less processing'...
    ' to do afterwards. If you want a good image to centre on, run Preview for'...
    ' a while till a good image is aquired. If your centre is far off, click the'...
    ' ''Centre'' button then click the image where you think the centre is. The Graph'...
    ' at the side will show you the intensity along cuts from the centre to'...
    ' the edge along the black lines. You can make small adjustments using'...
    ' the arrow buttons. If you set the centre near the edge of the camera'...
    ' image, the image will automatically crop to keep the picture square'...
    ' be careful not to crop out useful data this way - you may need to'...
    ' move the camera. If you wish to crop the image smaller yourself,'...
    ' set the width in pixels manually in the edit box.']);

elseif strcmp(title,'Preview')
```

```
helptxt=sprintf(['Live Camera View:\n\n'...
    'The Live Camera View button gives you the live camera feed, and a graph of the'...
    ' total signal vs time. Pausing the preview gives you access to the camera'...
    ' setting, which you can change to alter the preview image. Changing the'...
    ' main acquire time and/or framerate changes the number of pictures that are'...
    ' averaged to get the signal vs time graph.']);

elseif strcmp(title,'Naming Convention')
helptxt=sprintf(['File name convention: \n\n121125-03-11.mat:\nData from the '...
    '11th scan af the third run of 25/11/12\nDate (yy mm dd)\n - \nRun number (03)'...
    ' \n -\nScan no. (11) \nFormat (.mat)\n\nEach file contans all the pump-'...
    'probe and background images for that scan. In addition to this, each run '...
    'creates a settings file in the format 121125-03settings.mat. This file '...
    'contains information about the delay times each image was taken at, the '...
    'number of scans taken, the size of the data images and the time taken to '...
    'aquire each image. The program Process_Data uses the information in the settings '...
    'file to load all of the data files in a Run, sum them together, and do some '...
    'preliminary processing']);

elseif strcmp(title,'About')
helptxt=['This software was written by Ruth Livingstone over the course of'...
    ' her PhD. It is designed to be used with an imageingsource camera, two'...
    ' thorlabs shutters, and a PI stage. If any of these are replaced, the'...
    ' code will need to be altered to put the new serial numbers in (see'...
    ' Programming Help). It was written using matlab 7.4.0(R2007a)'];

elseif strcmp(title,'Programming')
helptxt=['Programming help is located at the beginning of the file'...
    ' "' file '.m" All the user interface (UI) and variable information'...
    ' is stored there. along with information about changing hardware and'...
    ' altering the user interface. To edit the user interface, type "guide"'...
    ' into the command line, and open "' file '.fig" when prompted.'];

elseif strcmp(title,'Camera Settings Changed')
helptxt=sprintf(['The Camera Settings have been successfully Changed!\n\n'...
    ' You may need to alter the Image Threshold. Do this by using'...
    ' ''Live View'' and the threshold slider to decide what the best value is']);

else helptxt=[title ' help information still to be written. Sorry!'];
end

helpdlg(helptxt,[file ' ' title]);%make the help window

%% The End!

% Old Picture Function (creates picture and graph and saves data)
% function picdisp(rdata,handles,setting,textdisplay,iset)
% global spec delays PumpPlot ProbePlot PumpProbe cdata hI hG Pe Pm Pp Pd Ps...
%     rect api scan Dataa Datab Datac imthresh csat sccomp centon
%
% %rdata      = raw data to be made into cropped picture and graph
% %setting    = Value to change output:
% %                  -1 (gaussian fit button)
% %                   0 (something in 'picture' buttongroup),
% %                   1 (Main Picture)
% %                   2 (Pump only Picture),or
% %                   3 (Probe only Picture)
% %                   9 (Run Aborted)
% %                  10 (Set up pics for first time)
% %textdisplay= string to display in window
% %handles    = all the variables and values
%
% %get and calculate variables
% % load dataf                                %wd is saved in dataf
% % load piccentre
% x    = handles.value.piccentre(1);           %x value of centre
% y    = handles.value.piccentre(2);           %y value of centre
% w    = handles.value.picwidth;               %width is the shortest distance from the centre to the edge
```

```
% cdata= rdata(y-w+1:y+w-1,x-w+1:x+w-1,:);        %crop picture
% cdata= centroid(cdata,imthresh,centon);         %subtract camera noise and centroid image, as requested
% lw   = min([floor(w/2) 8]);                       %line width
% up   = mean(cdata(w:-1:1, w-lw:w+lw),2);      %blue(mean(...,2)=avg the rows)
% down = mean(cdata(w:(2*w-1),w-lw:w+lw),2);    %green
% left = mean(cdata(w-lw:w+lw,w:-1:1),1);       %red(mean(...,1)=avg the columns)
% right= mean(cdata(w-lw:w+lw,w:(2*w-1)),1);    %turquose
% summ = (left+right+transpose(up+down))/4;
% inten= mean(mean(cdata(:,:)));
% if get(handles.piccross,'Value'),cdata(:,w-1:w+1)=NaN; cdata(w-1:w+1,:)=NaN;end     %make centre cross
%
% %if Run button pushed, set up variables to be saved
% if setting ~=0
%     sett2=iset+length(delays)*sccomp;
%     DataI=int8(ceil(cdata)-128);
% end
%
% %Create and display Graphs:
% if setting==0                                   %something in 'picture' buttongroup
%     axes(handles.Graph)
%     plot(1:w,up,1:w,down,1:w,left,1:w,right,1:w,summ);%plot graph
%     xlabel('distance from centre (pixels)'); ylabel('Intensity(%)')
%     title('Energy Distribution')
%     legend('up','down','left','right','average','Location','NorthOutside')%create key
%     set(handles.Graph,'YTick',[25 50 75 100])   %set the tickmarks
%     axis([0 w 0 100])                           %set the axes limits [xmin xmax ymin ymax]
%     textdisplay=[textdisplay 'Centre: (' num2str(x) ', ' num2str(y) ')']; %display centrepoint
%     set(handles.piccentretxt, 'String', [num2str(x) ', ' num2str(y)])
%     axes(handles.Picture);pcolor(cdata);colormap jet  %Display the picture
%     shading interp;axis equal tight off;view(handles.Picture,0,90)
% elseif setting==10                              %Initial Set up
%     if iset==length(delays) %End of scan
%         disp(['scan ' num2str(scan) ' saved as:' textdisplay])
%         save(textdisplay,'Dataa','Datab','Datac')%SAVE SCAN DATA.
%     end
% %Set up variables
%     scannum = handles.value.scannum;
%     framerate= handles.value.framerate;
%     shotspp = round(handles.value.imagetime*framerate); if shotspp==0, shotspp=1; end
%     shotsb = round(handles.value.btime*framerate); if shotsb==0, shotsb=1; end
%     shotsc = round(handles.value.ctime*framerate); if shotsc==0, shotsc=1; end
%     if handles.value.backsub, shotsb=0;shotsc=0;end
%     i=size(cdata);
%     Data=zeros(i(1),i(2),length(delays),'int8');%reset everything to zero and asign space to large variables
%     Dataa=Data;Datab=Data;Datac=Data;
% %SET UP GRAPHS
% %set up spectra/delays graph
%     axes(handles.Graph)
%     spec= zeros(length(delays),w);     %reset values
%     hG  = surf(1:w,delays,spec,spec,'CDataSource','spec');%plot Graph
%     view(handles.Graph,0,90),shading flat,colormap jet, axis tight
%     rect= [1 delays(1) w-2 delays(end)-delays(1)];             %set up selection rectangle
%     h2  = imrect(handles.Graph,rect);
%     api = getappdata(h2,'API');
%     fcn = makeConstrainToRectFcn('imrect',get(handles.Graph,'XLim'),get(handles.Graph,'YLim'));
%     api.setDragConstraintFcn(fcn);
%     api.addNewPositionCallback(@RectFn)
% %set up camera image graph
%     axes(handles.Picture);
%     x=size(cdata,1);y=size(cdata,2);               %reset values
%     hI=surf(1:y,1:x,zeros(x,y),cdata,'CDataSource','cdata');%Display the picture
%     shading interp; axis equal tight off; view(handles.Picture,0,90)
% %set up the background intensity graphs
%     a=zeros(1,length(delays)*handles.value.scannum);
%     PumpPlot=a; ProbePlot=a; PumpProbe=a; b=1:length(a);
%     axes(handles.ProbePlot),Pe=plot(b,ProbePlot,'YDataSource','ProbePlot');axis off
%     axes(handles.PumpPlot), Pm=plot(b,PumpPlot ,'YDataSource','PumpPlot' );axis off
%     axes(handles.PumpProbe),Pp=plot(b,PumpProbe,'YDataSource','PumpProbe');axis off
```

```
% %set up delays and spectra sidegraphs
%     axes(handles.Delays),cla(handles.Delays)
%     Pd=plot(delays,delays,'XDataSource','mean(trimspec,2)','YDataSource','x');
%     set(Pd,'HandleVisibility','off')               %this is so I can plot gaussian fit later
%     axes(handles.Spectra),Ps=plot(1:w,1:w,'XDataSource','rw1:rw2','YDataSource','mean(trimspec,1)');
%     ylim(handles.Delays, [delays(1) delays(end)]), xlim(handles.Spectra,[1 w])
% %Save settings file
%     Isize=size(cdata);
%     save(textdisplay,'delays','shotspp','shotsb','shotsc','scannum','Isize')
% elseif setting==1||setting==-1                   %Main Picture
%     if setting==1
%         spec(iset-1,:)=spec(iset-1,:)+summ;
%         PumpProbe(sett2-1:end)=inten;
%         Dataa(:,:,iset-1)=DataI;                 %setup data for file
%         refreshdata(hI,'caller')                %display Image
%     end
%     rect=api.getPosition();                      %find out where the draggable rectangle is
%     r1=find(delays<rect(2));r2=find(delays>rect(4)+rect(2));%put that in context of the graph
%     if isempty(r1),rdel1=1;else rdel1=r1(end)+1;end
%     if isempty(r2),rdel2=length(delays);else rdel2=r2(1)-1;end
%     x=delays(rdel1:rdel2)';
%     rw1=round(rect(1));rw2=round(rect(3)+rect(1));
%     trimspec=spec(rdel1:rdel2,rw1:rw2);          %crop the sidegraph data
%     refreshdata(hG,'caller')                     %plot Graph
%     refreshdata(Pp,'caller')                     %plot intensity
%     refreshdata(Ps,'caller')                     %plot spectra sidegraph
%     refreshdata(Pd,'caller')                     %plot delays sidegraph
%     gfit=1;if setting==1 && (iset-2)~=(length(delays)-1), gfit=0;end %only fit gaussin if end of scan
%     if gfit,y=tracefit(x,mean(trimspec,2),handles);
%         if ~isempty(y)
%             axes(handles.Delays),cla(gca)         %clear last fit line
%             refreshdata(Pd,'caller'),hold on
%             plot(handles.Delays,y,x,'-r'),hold off %plot new fit line
%         end
%     end
%     if (iset-1)==length(delays)
%         disp(['scan ' num2str(scan-1) ' finished at ' datestr(now,16) ', saved as: ' textdisplay])
%           save(textdisplay,'Dataa','Datab','Datac')%SAVE SCAN DATA.
%     end
%     if csat
%         set(handles.csat,'Visible','on')
%         fid=fopen([textdisplay(1:end-6) '.txt'],'a');%WRITE TO A FILE
%         csattxt=['Camara Saturated on scan no. ' num2str(scan) ' and delay time' num2str(delays(iset-1))];
%         errordlg(csattxt)
%         fprintf(fid,csattxt);
%         fclose(fid)
%         handles.value.csatfile=[textdisplay(1:end-6) '.txt'];
%     end
% elseif setting==2                                %Pump only Picture
%     PumpPlot(sett2:end)=inten;                   %add data to pump background plot vector
%     refreshdata(Pm,'caller')                     %plot pump background graph
%     Datab(:,:,iset)=DataI;                       %setup data for file
% elseif setting==3                                %Probe only Picture
%     ProbePlot(sett2:end)=inten;                  %add data to probe background plot vector
%     refreshdata(Pe,'caller')                     %plot probe background graph
%     Datac(:,:,iset)=DataI;                       %setup data for file
% elseif setting==9                                %Scan Aborted, save incomplete scan.
%     disp(['incomplete scan ' num2str(scan) ' saved as:' textdisplay])
%     save(textdisplay,'Dataa','Datab','Datac')%SAVE SCAN DATA.
% end
%
% % if setting ==3 && iset==length(delays)         %End of scan
% % end
% inten=['Intensity = ' num2str(inten,3) '%    '];%Calculate intensity
% set(handles.textdisplay,'String',[inten textdisplay])%Display the text
% handles.value.picwidth=w;                       %Save new picwidth
% set(handles.picwidth,'String',handles.value.picwidth*2); guidata(handles.picwidth,handles)
% drawnow
```

Appendix.C. Process Data

```
function varargout = Process_Data_1_0(varargin)
% Processes Raw Data taken by Aquire_Data_1_0

%Programming Help

% Last Modified by GUIDE v2.5 01-May-2012 10:13:40

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Process_Data_1_0_OpeningFcn, ...
                   'gui_OutputFcn',  @Process_Data_1_0_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
%% Setup Figure
% --- Executes just before Process_Data_1_0 is made visible.
function Process_Data_1_0_OpeningFcn(hObject, eventdata, handles, varargin)
global api file
%setup handles structure
handles.output = hObject;
%if the program is being opened by another program, wanting to open a
%specific file, i.e. Process_Data_1_0('file',filename)
file=getpath;
if ~isempty(varargin),file=varargin{end};end
%setup scrollbox
set(handles.ProcessData,'Renderer','zbuffer')
hIm = imshow(magic(5),'Parent',handles.Images);         %use a standard picture for setting up
hSP = imscrollpanel(handles.uipanel2,hIm);              %setup scrollbox with this image
api = iptgetapi(hSP);                                   %setup api so that scrollbox settings can be modified
handles.txt=text(0,0,'','Parent',handles.Images);       %setup text handle
handles.settings.settfile=file;
handles.settings.timestamp=[];
guidata(hObject,handles)                                %save
loadsettdata(handles,file,1)

% --- Outputs from this function are returned to the command line
function varargout=Process_Data_1_0_OutputFcn(hObject, eventdata, handles)
varargout{1}=handles.output;
% set figure colormap
load MyColormaps/ProcessDefault.mat
set(handles.ProcessData,'ColorMap',cmp)

%% Add Data File
% add image function (adds an image or a background image to an image file)
function addimage(handles,Q)
scale=char(inputdlg(sprintf('What Scaling factor do you want to apply to your new image?\n\nScale:'),'',1,{'1'}));
if ~isempty(scale) && ~isnan(str2double(scale)); loadsettdata(handles,scale,Q), end

%add data function (more general function)
function adddatafile(handles)
[ofile nfile]=getpath;
if strcmpi(handles.settings.ftype,'Image'),addimage(handles,2),return,end
loadsettdata(handles,nfile,2)
handles=guidata(handles.xx);%update handles
```

```
Exit=0;
try
    if ~isequal(handles.settings2.delays,handles.settings.delays)
        OK=questdlg('Please choose files with the same number and position of time steps','','OK','Cancel','OK');
        if strcmp(OK,'OK'),adddatafile(handles)
        else,return,end
    end
catch Exit=1; %if image files
end
if Exit,return,end
handles=guidata(handles.xx);%update handles
c={num2str(handles.settings2.piccentre(2)),num2str(handles.settings2.piccentre(1)),['[1:'
num2str(handles.settings2.scannum) ']']};
answer = inputdlg({sprintf('Enter Centrepoint and scanrange for new Dataset:\n\nx');'y';'scan'},'',1,c);
c2=str2double(answer)';
if isempty(c2),return,end
a=cell2struct(answer(3),'b');
scans2=eval(a.b);
wd= handles.settings.picwidth;                          %first picture width
c1= handles.settings.piccentre;
W = handles.settings2.picwidth*2;                        %old width
w = min([c2(1) c2(2) W-c2(1) W-c2(2) wd]);               %calculate new width
handles.settings2.picwidth = w;
handles.settings.picwidth = w;
handles.settings2.piccentre= c;
cy1=c1(1)-w+1:c1(1)+w; cx1=c1(2)-w+1:c1(2)+w;            %decide what data 1 to keep
cy2=c2(1)-w+1:c2(1)+w; cx2=c2(2)-w+1:c2(2)+w;            %decide what data 2 to keep
b1=handles.settings.shotsb;  c1=handles.settings.shotsc;  pp1=handles.settings.shotspp;
b2=handles.settings2.shotsb; c2=handles.settings2.shotsc; pp2=handles.settings2.shotspp;

[path2 name2    ]=fileparts(handles.settings2.settfile);
[path1 name1 ext]=fileparts(handles.settings.settfile);
name=[name1(1:end-8) '+' name2(1:end-8) 'settings' ext];
[name path]=uiputfile('*.mat','Save added file as',[path1 '/' name]);
name=name(1:end-12);                                     %get rid of 'settings.mat' from end of name
sc=length(handles.settings.scans);
shotspp=1; shotsb=1; shotsc=1; Isize =[w*2 w*2]; delays=handles.settings.delays;  %#ok<NASGU>
scannum=length(scans2)+sc;
h=waitbar(0.01,'Cropping Data and Saving to Added file...');
for i=handles.settings.scans
    waitbar(i/(scannum+1))
    load([handles.settings.settfile(1:end-12) '-' num2str(i) '.mat'])
    Dataa=single(Dataa(cx1,cy1,:)+128)/pp1-128;
    Datab=single(Datab(cx1,cy1,:)+128)/b1-128;
    Datac=single(Datac(cx1,cy1,:)+128)/c1-128;
    save([path '/' name '-' num2str(i) '.mat'],'Dataa','Datab','Datac')
end
for j=scans2
    i=i+1; waitbar((i)/(scannum+1))
    try
        load([handles.settings2.settfile(1:end-12) '-' num2str(j) '.mat'])
        Dataa=single(Dataa(cx2,cy2,:)+128)/pp2-128;
        Datab=single(Datab(cx2,cy2,:)+128)/b2-128;
        Datac=single(Datac(cx2,cy2,:)+128)/c2-128;
        save([path '/' name '-' num2str(i) '.mat'],'Dataa','Datab','Datac')
    catch scannum=i-1+sc; break,            %#ok<NASGU>
    end                     %if there are less scan files than expected (scan aborted etc), don't panic.
end
waitbar(0.9)
save([path name 'settings' ext],'delays','scannum','shotspp','shotsb','shotsc','Isize')
close(h)
guidata(handles.xx,handles) %save values
loadsettdata(handles,[path '/' name 'settings' ext],1)

l = sc;                                  %find out the point when one run ends and the next one starts
m = length(handles.settings.delays);     %find out how many pics there are
n = handles.settings.scannum;            %find out how many scans there are
A = reshape(handles.Data.inteny,m,n);    %Turn the scantrace into a matrix
```

```
pl= @(a,b,l,A) plot(1:size(A,1),mean(A(:,1:l),2)*str2double(get(a,'String')),1:size(A,1),mean(A(:,l:end),2));
figure,h = uicontrol('Style','Edit','String','1','Callback',{pl,l,A});%make the editbox
pl(h,h,l,A)                                  %plot the graph

%% Load Settings File
function loadsettdata(handles,settfile,Q)
%handles:   structure of variables and UI handles.
%settfile:  empty:    (no file chosen yet. Let user pick one)
%           filename: (filename of file that will be loaded including path and extension.
%                      e.g.'C:\Users\DB2.27\Documents\MATLAB\Test Lab Data\NOsettings.mat')
%Q:         1 - first file, 2 - added file
Title='Pick a settings file';I=1;
[path Good Image]=getpath; name='';
scale = str2double(settfile); if isnan(scale),scale=1;end %settfile can sometimes be used to pass a scaling factor
(see function addimage)
if~isempty(settfile),[path name]=fileparts(settfile);end
filetype='Scan';
%get file to load
if ~isempty(name)                                        %check if you already know the file
    try settings=load(settfile);                         %check if the filename is valid
        if isfield(settings,'shotspp'),filetype='Scan';
        else filetype='Image';end
    catch name='';end                                       %make name empty so that next "if" will know that no valid
file exists
end
if isempty(name)                                         %check again if you don't already know the file
    if   Q==1;   filetype=questdlg('Do   you   want   to   open   a   scan   file   or   a   single   image?','File
Type','Scan','Image','Cancel','Scan');
    else filetype=handles.settings.ftype; end
    if strcmpi(filetype,'Cancel'),return
    elseif strcmpi(filetype,'Image'),
        name='*.mat'; path=Image; Title=[Title ' (image files should be grayscale)'];
        ftp={'*.mat','Load Raw Data (*.mat)';'*.tif;*.bmp;*.jpg','Image file (*.tif;*.bmp;*.jpg)'};
    elseif strcmpi(filetype,'Scan'),
        name='*settings.mat'; ftp={'*.mat','Load Raw Data (*.mat)'};
    end
    [filename, pathname, I]=uigetfile(ftp,Title,[path '\' name]); %open a "open file" window
    if isequal(filename, 0)                              %if the user presses "cancel" on the open file window
        set(handles.displaytxt,'String',sprintf('New File not loaded'))
        return                                           %go back to the main program
    else settfile=[pathname filename];if I==1,settings=load(settfile);end%get the new filename and load file
    end
end
settings.ftype    = filetype;
settings.settfile = settfile;
if strcmpi(filetype,'Scan')
    %calculate the settings
    %Settings file contains:delays,scannum,shotsb,shotsc,shotspp,Isize (and sometimes piccentre)
    set(handles.ImageSubtract,'Enable','off')
    if isfield(settings,'shotsbg'),settings.shotsb=settings.shotsbg; settings.shotsc=settings.shotsbg; end
    if isfield(settings,'Isize'),  settings.size = settings.Isize; end
    settings.scans    = 1:settings.scannum;
    w = floor(settings.size(1)/2);
    settings.picwidth = w;
    if ~isfield(settings,'piccentre'),settings.piccentre=[w w];end
    del = settings.delays;
    a   = cell(length(del),1);                           %set up the cell array to put text strings in
    for i=1:length(del),a{i}=num2str(del(i),'% 6.0f');end   %create text strings from "delays" and put in the rest of
the array
    set(handles.delays,  'String', a)                    %put these strings in the delays listbox
    set(handles.scans,   'String',['[1:' num2str(settings.scannum) ']'])%update 'Choose Scans' Editbox
    set(handles.picwidth,'String', num2str(settings.picwidth*2))%update 'Width' Editbox
    try delete(handles.txt),end                          %if there is old text, delete it.
    handles.txt=text(0,0,'','Parent',handles.Images);    %setup text handle again
    settings.trmin=min(settings.delays);    settings.trmax=max(settings.delays);
    set(handles.trmin,    'String', num2str(min(settings.delays)));
    set(handles.trmax,    'String', num2str(max(settings.delays)));
    if Q==1,
```

```
        handles.settings=settings;
        guidata(handles.xx,handles)                         %save settings
        loaddata(handles,settfile)                          %if it does, load the scan data files
    else
        handles.settings2=settings;
        guidata(handles.xx,handles)                         %save settings2
    end

else %image file
    %standard file contains:data sdata timestamp c
    %is also able to open any *.mat file containing a matrix greater than 20x20
    settings.scannum=1; settings.shotspp=1;    settings.scans =1;
    settings.shotsb =1; settings.delays =1000; settings.shotsc=1;
    settings.trmin  =1; settings.trmax  =1;
    handles.Data.Trace = 1;   handles.Data.inteny= [];   handles.Data.intenx= [];
    set(handles.ImageSubtract,'Enable','on')
    if I==1;    %if matlab *.mat file
        try %try opening a standard picture file
            setting.size=fliplr(size(settings.data));
            settings.picwidth    = settings.c.w;
            settings.piccentre(1)= settings.c.lr;
            settings.piccentre(2)= settings.c.ud;
        catch %if the image file is not of your standard setup
            %extract data
            fields=fieldnames(settings); c=zeros(length(fields),4);
            for i=1:length(fields), %see what you have in your file
                [c(i,1)  c(i,2)  c(i,3)]=size(getfield(settings,char(fields(i))));       %find out the size of each
variable
                c(i,4)=ischar(getfield(settings,char(fields(i))));  %find out if any of the variables are text
            end
            %get the data
            pic=find(c(:,1)>20 & c(:,2)>20); %see if there are any datasets bigger than 20x20 in the file
            if any(pic),settings.data = getfield(settings,char(fields(pic(1))));%save that dataset as your picture
            else errordlg('No Picture Data found in file'),uiwait,return,end    %if not, otherwise, throw an error
            %get any text
            if any(c(:,4)),settings.timestamp = getfield(settings,char(fields(find(c(:,4),1)))); %extract the first
text field from the file
            else settings.timestamp   = 'single image'; end
            %save stuff
            settings.picwidth    = floor(size(settings.data,1)/2);
            settings.piccentre(1)= settings.picwidth;
            settings.piccentre(2)= settings.picwidth;
            %see if it's an image file or a scan file
            if length(pic)>1
                %see how many of the variables are the same size as the first picture
                for i=2:length(pic), good(i)=all((c(pic(i),1:2)==c(pic(1),1:2))); end
                good=find(good);
                for i=1:length(good)%add the pictures together
                    settings.data(:,:,end+1:end+c(pic(good(i))),3) = getfield(settings,char(fields(pic(good(i)))));
                end
            end
            d=size(settings.data,3);
            if d>1 %if the file has more than one picture in it
    %             settings.ftype  = 'Scan';
                % see if the file has a list of time values
                isd=find(any(c(:,1:2)==d,2)); %check if it has a variable that is the same length as the number of
pictures
                if any(isd), settings.delays = getfield(settings,char(fields(isd(1)))); %if it does, save the first
one it finds
                else settings.delays=(1:d)*1000; end %if it doesn't, make up something
            end
        end
    else        %if image from image file (tif, bmp or jpg)
        settings.data        = double(sum(imread(settfile),3));
        settings.picwidth    = floor(size(settings.data,1)/2);
        settings.piccentre(1)= settings.picwidth;
        settings.piccentre(2)= settings.picwidth;
        settings.timestamp   = 'single image';
```

```
    end

    rData = settings.data;  settings.size = size(settings.data);
    cData = zeros(settings.size);   bData = zeros(settings.size);
    if Q==1                                                 %first file
        handles.Data.rData = rData;
        handles.Data.bData = bData;
        handles.Data.cData = cData;
        handles.settings   = settings;
    elseif Q==2                                             %second image file
        c1= handles.settings.piccentre; c2= settings.piccentre;
        if any(c1~=c2)
            w = min([c1 c2 640-[c1(2) c2(2)] 480-[c1(1) c2(1)]]);   %calculate new width
            cx1=c1(1)-w+1:c1(1)+w; cy1=c1(2)-w+1:c1(2)+w;           %decide what data 1 to keep
            cx2=c2(1)-w+1:c2(1)+w; cy2=c2(2)-w+1:c2(2)+w;           %decide what data 2 to keep
            handles.settings.piccentre    = [w w];
        else cy1=1:640; cy2=1:640; cx1=1:480; cx2=1:480;
        end
        handles.Data.rData = handles.Data.rData(cx1,cy1,:);
        handles.Data.bData = handles.Data.bData(cx1,cy1,:);
        handles.Data.cData = handles.Data.cData(cx1,cy1,:);
        handles.Data.rData(:,:,end+1) = scale*rData(cx2,cy2);
        handles.Data.bData(:,:,end+1) = scale*bData(cx2,cy2);
        handles.Data.cData(:,:,end+1) = scale*cData(cx2,cy2);
        handles.settings.delays(end+1)= (length(handles.settings.delays)+1)*1000;
    elseif Q==3                                             %background image file
        for i=1:size(handles.Data.rData,3),bData(:,:,i)=rData; end
        handles.Data.bData = scale*bData;
    end
    guidata(handles.pcentreUP,handles)                      %save data
    del=handles.settings.delays;
    for i=1:length(del),a{i}=num2str(del(i),'% 6.0f');end   %create text strings from "delays" and put in the rest of
the array
    set(handles.delays, 'String', a, 'Value',1)             %put these strings in the delays listbox
    picdisp(handles,0,settings.timestamp)                   %go to 'Picture Function'
    graphs = [handles.ScanTrace handles.Trace];               %get the graphs you want to turn invisible (not used in
this mode)
    set([graphs;get(graphs(1),'Children'),get(graphs(2),'Children')],'Visible','off')%turn them and all the data on
them invisible
end

set(handles.ProcessData, 'Name',   ['Process Data: ' settfile])
set(handles.picwidth,'String', num2str(settings.picwidth*2))
set(handles.xx,      'String', num2str(settings.piccentre(2)))
set(handles.yy,      'String', num2str(settings.piccentre(1)))
%% Load Data Files
function loaddata(handles,settfile)
global rData bData cData
h  = waitbar(0,'Loading...','CreateCancelBtn','delete(gcbf)'); %set up waitbar
pic= length(handles.settings.delays);                   %number of steps per scan
sc = handles.settings.scans;                            %which scans have been chosen (i.e. [10:14,20:40,55])
b  = 0;
a  = linspace(0,1,pic);                                     %a is used to help create scale for scantrace,b is a
loopcounter
handles.Data.inteny = []; handles.Data.intenx = [];
handles.Data.Trace  = zeros(1,pic);
try D = load([settfile(1:end-12) '-1.mat']);
catch
    errordlg('This file does not currently contain any scan data. File will not be opened')
    err=lasterror;err.message,err.stack
    return
end
handles.Data.rData=single(D.Dataa-D.Dataa);                 %assign space for raw Data
handles.Data.bData=single(D.Dataa-D.Dataa);                 %assign space for background Data
handles.Data.cData=single(D.Dataa-D.Dataa);                 %assign space for background Data
for i=sc                                                %for all the chosen scans
    b=b+1;                                              %count the loop
    if ~ishandle(h), return, end                        %exit if the user has closed the waitbar
```

```
    waitbar(b/(length(sc)+2));                          %update the waitbar
    try D=load([settfile(1:end-12) '-' num2str(i) '.mat']);%load the correct scan file
    catch                                               %if there are less scan files than expected (scan aborted
etc), don't panic.
        sc=1:i-1;handles.settings.scannum=i-1;handles.settings.scans=1:i-1;
        set(handles.scans,'String',['[1:' num2str(handles.settings.scannum) ']'])
        break
    end
%     [m x]=max(D.Datac(:,:,1));[m y]=max(m);x=x(y);    %USE THESE 3 LINES IF CAMERA NOISE IS A PROBLEM
%     D.Dataa(x,y,:)=-128; Dataa=D.Dataa; D.Datab(x,y,:)=-128; Datab=D.Datab; D.Datac(x,y,:)=-128; Datac=D.Datac;
%     save([settfile(1:end-12) '-' num2str(i) '.mat'],'Dataa','Datab','Datac')
    for x=1:pic,                                        %for each scan step in that file
        handles.Data.inteny(x+pic*(b-1))=mean(mean((D.Dataa(:,:,x)))+128;%get the average pump-probe intensity
        handles.Data.intenx(x+pic*(b-1))=i+a(x)-1;      %make the scale to plot intensity against
        handles.Data.rData(:,:,x)=handles.Data.rData(:,:,x)+single(D.Dataa(:,:,x))+128;%add the raw data
        handles.Data.Trace(x) = handles.Data.Trace(x)+mean(mean((D.Dataa(:,:,x))))+128;
        if handles.settings.shotsb~=0;                  %if there are background files
            handles.Data.bData(:,:,x)=handles.Data.bData(:,:,x)+single(D.Datab(:,:,x))+128;
            handles.Data.cData(:,:,x)=handles.Data.cData(:,:,x)+single(D.Datac(:,:,x))+128;
        end
    end
end
%get rid of faulty data points
handles.Data.rData(isnan(handles.Data.rData))=0;
handles.Data.bData(isnan(handles.Data.bData))=0;
handles.Data.cData(isnan(handles.Data.cData))=0;
handles.Data.rData(isinf(handles.Data.rData))=0;
handles.Data.bData(isinf(handles.Data.bData))=0;
handles.Data.cData(isinf(handles.Data.cData))=0;

waitbar(length(sc)+1/(length(sc)+2));               %update the waitbar
xs=handles.settings.size(1);ys=handles.settings.size(2);
for x=1:xs
    for y=1:ys
        r=sqrt((x-xs/2)^2+(y-ys/2)^2);              %calculate radius
        handles.Data.rad(x,y,1)=r;                  %store radius
        a=atan(sqrt(y-ys/2)^2/(x-xs/2)^2);          %calculate angle
        handles.Data.rad(x,y,2)=a*180/pi;           %store angle
    end
end
fclose('all')
close(h)                                            %close the waitbar
handles.settings.settfile=settfile;
rData=handles.Data.rData;bData=handles.Data.bData;cData=handles.Data.cData;
guidata(handles.xx,handles)                         %save the data
[path name]=fileparts(settfile);
ttext=['Data File: ' name(1:end-8) '...mat' sprintf('\nContains ') ...%create the display String
    num2str(handles.settings.scannum) ' scans, with ' num2str(pic) ' steps per scan'];
picdisp(handles,0,ttext)                                %go to 'Picture Function'

%% Process Pictures (Abel and Symetrise)

function Data=picproces(Data,ww,shots,sym,abel,pol)
%pol=1: polar image out.
%pol=0: cartesian image out.
w   = ww(3);
cy  = ww(2)-w+1:ww(2)+w;  cx=ww(1)-w+1:ww(1)+w;         %decide what data to keep
Data= Data(cy,cx,:)/shots;
% Data= double(Data(cy,cx,:))/shots;
if sym                                                  %if symmetrize ticked
    Data = Data(w+1:2*w,w+1:2*w,:)+Data(w:-1:1,w:-1:1,:)+Data(w+1:2*w,w:-1:1,:)+Data(w:-1:1,w+1:2*w,:);
    if abel, Data=abelt(Data,1); end %get abel transform of just 1/4 of image (faster)
    Data(1:w,1:w,:)       = Data(w:-1:1,w:-1:1,:);
    Data(1:w,2*w:-1:w+1,:)= Data(1:w,1:w,:);
    Data(2*w:-1:w+1,:,:)  = Data(1:w,:,:);
%     Data=[fliplr(Data) Data; flipud(fliplr(Data)) flipud(Data)];
end
if abel&&~sym,
```

```
    Data=abelt(Data,0);                              %if Abel ticked
%     oData=abelb(oData,'lsqnonneg');               %use ben's code
end


%{
USE THIS LINE TO SYMATRISE IN MATLAB WINDOW.
Name image data 'datap'. Define centrepoint using c1 & c2, define colormap limits using contrast.
c1=263;c2=317;w=200;contrast=2;D=datap(c1-w:c1+w,c2-w:c2+w);rD=D(w+1:2*w,w+1:2*w,:)+D(w:-1:1,w:-1:1,:)+D(w+1:2*w,w:-
1:1,:)+D(w:-1:1,w+1:2*w,:);D(1:w,1:w,:)=rD(w:-1:1,w:-1:1,:);D(1:w,2*w:-1:w+1,:)=D(1:w,1:w,:);D(2*w:-
1:w+1,:,:)=D(1:w,:,:);imshow(D,[min(min(D)) max(max(D))*contrast]),colormap jet
%}

%Abel Transform Data
function Data=abelt(Data,quart)
w = size(Data,2)/2;
if quart, w=w*2; end
A = AbelA(w); A1=inv(A);                                    %create the inverse Area Matrix
Data=permute(Data,[2 1 3]);%display images sideways
for ll=1:size(Data,3)
    if quart, Data(:,:,ll) = 0.5*A1*Data(:,:,ll);
    else
        Data(w:-1:1,:,ll)  = 0.5*A1*Data(w:-1:1,:,ll);          %transform top half of raw image
        Data(w+1:2*w,:,ll) = 0.5*A1*Data(w+1:2*w,:,ll);         %transform bottom half of raw image
    end
end
Data=permute(Data,[2 1 3]);

%Abel Transform Data using Ben Sussman's code
function Data=abelb(Data,method)
for ll=1:size(Data,3)
    [Ir, r, Data(:,:,ll)]=VMIabelinv2(Data(:,:,ll),method);
end

%% Picture Function
function picdisp(handles,setting,textdisplay)
global api data beta rD
%get Values
scl   = get(handles.Scl,'Value');
hIms  = handles.Images;         y     = handles.settings.piccentre(1);
hIm   = handles.Im;             x     = handles.settings.piccentre(2);
hSpec = handles.Spectra;        w     = handles.settings.picwidth;
hScan = handles.ScanTrace;      bon   = get(handles.betaon,'Value');
mycmap= get(gcf,'Colormap');    abel  = get(handles.AbelTranform,'Value')||get(handles.halfimage,'Value');
hTrace= handles.Trace;          sym   = get(handles.symmetrize,'Value');
fh    = handles.ProcessData;        delays= handles.settings.delays;
BS    = strcmpi(get(handles.BackShow,'Checked'),'on');      %find out if we want to see the background images
SS    = strcmpi(get(handles.SelectShow,'Checked'),'on');    %find out if we only want to see some images
i     = get(handles.delays,'Value');                        %get which image has been selected from the listbox
[path nm] = fileparts(handles.settings.settfile(1:end-12));
if SS, ii = i; else, ii = 1:length(delays); end                   %choose which images to show
li    = length(ii);
if get(handles.halfimage,'Value');
    vis = 'on';  sc = (1-get(handles.hascale,'Value'))*2;
else
    vis = 'off'; sc = 1;
end
set([handles.hascale handles.hatxt],'Visible',vis)
%setup exporting figure if neccesary
if setting,
    fh=figure;
    switch setting
        case 1;hIms  = gca; name='Images';  sz=[1.1 ceil(li/10)/7];
        case 2;hIm   = gca; name='Image';   sz=1;
        case 3;hSpec = gca; name='Spectrum';sz=0.5;
        case 4;hScan = gca; name='Scan';    sz=0.5;
        case 5;hTrace= gca; name='Trace';   sz=0.5;
    end
```

```
end
rD = handles.Data.rData(:,:,ii)/handles.settings.shotspp;
bD = handles.Data.bData(:,:,ii)/handles.settings.shotsb;
cD = handles.Data.cData(:,:,ii)/handles.settings.shotsc;
% %UNCOMMENT THIS IF YOU WANT TO PERFORM ABEL TRANSFORMS BEFORE SUBTRACTING (DOESN'T WORK FOR 1/2 ABEL)
% rD = picproces(handles.Data.rData(:,:,ii),[x y w],handles.settings.shotspp,sym,abel,0);
% bD = picproces(handles.Data.bData(:,:,ii),[x y w],handles.settings.shotsb,sym,abel,0);
% cD = picproces(handles.Data.cData(:,:,ii),[x y w],handles.settings.shotsc,sym,abel,0);

try
%subtract background
    if get(handles.pump,'Value'),rD=rD-bD;end
    if get(handles.probe,'Value'),rD=rD-cD;end
%get raw data for half abel
    if get(handles.halfimage,'Value')
        rDD = picproces(rD,[x y w],1,sym,0,0);
    end
%process the data (symetrise and abel transform etc) REMOVE THIS IF YOU WANT TO PERFORM ABEL TRANSFORMS BEFORE
SUBTRACTING
    rD = picproces(rD,[x y w],1,sym,abel,0);
%normalise images
    mx = (max(max(max(rD(:,1:round(w*scl),:)))))*sc;
    rD=rD/mx; bD=bD/mx; cD=cD/mx; %normalise
    scale=[0 1];
%make half abel image
    if get(handles.halfimage,'Value')
        mx = max(max(max(rDD(:,1:round(w*scl),:))));
        rD(:,w-1:2*w,:)=rDD(:,w-1:2*w,:)/mx;
    end
%log scale images
    if get(handles.logimage,'Value'),
        rD(rD<0.001)=0.001;
        rD=real(log(rD));
        mn=min(min(min(rD(isfinite(rD)))));mx=(max(max(max(rD(:,1:round(w*scl),:)))));
        rD(~isfinite(rD))=mn; rD=(rD-mn)/(mx-mn);%normalise & get rid of dodgy values
        if BS,
            bD=log(bD); bD(~isfinite(bD))=mn; bD=(bD-mn)/(mx-mn);
            cD=log(cD); cD(~isfinite(cD))=mn; cD=(cD-mn)/(mx-mn);
        end
    end
    if SS,data=mean(rD,3);else,data=mean(rD(:,:,i),3);end
    [rInt beta rad] = IntegrateImage(data,2,w,180,3,bon,0,'no');%get polar image
%PLOT GRAPHS
%create the scrolling image
    try delete(handles.txt),catch,end                       %remove old text
    a = get(handles.delays,'String'); a=a(ii);              %get string array of delays values from listbox
    a = cellstr(strjust(char(a),'center'));                 %centre the strings (adds spaces to the string array)
    tx=w*0.8:w*2:w*2*length(a);ty=zeros(size(tx))-w/10;mg=55/w;  %decide the points to put the text(one point for each
picture,placed 1/20th of the pic width above)
    if w>50 && li>5                                 %if it's a big pic, make smaller (avoid OUT OF MEMORY error)
        rD=rD(1:2:end,1:2:end,:);
        if BS
            bD=bD(1:2:end,1:2:end,:);
            cD=cD(1:2:end,1:2:end,:);
        end
        tx=w*0.4:w:w*length(a);ty=zeros(size(tx))-w/20;mg=110/w;%decide the points to put the text(one point for each
picture,placed 1/40th of the pic width above)
    end
    if strcmp(handles.settings.ftype,'image'),mg=400/w;end
    RD=rD(:,:);                                             %make the data for the long thin image
    if BS && any(bD(:)), bD=bD(:,:); size(bD),size(RD), RD=cat(1,RD,bD);end
    if BS && any(cD(:)), cD=cD(:,:); RD=cat(1,RD,cD);end    %make 3 long thin images
    handles.txt=text(tx,ty,a,'Parent',hIms);                %write delays on image as text
    loc = api.getVisibleLocation();                         %get location of picture in scrollpanel
    api.replaceImage(RD,scale);                             %update image in scrollpanel
    api.setMagnification(mg);                               %set magnification in scrollpanel
    api.setVisibleLocation(loc);                            %set location to what it was before
%create the Scan Trace graph
```

```
    plot(handles.Data.intenx,handles.Data.inteny,'Parent',hScan);%plot graph
    xlim(hScan,[0,max(handles.settings.scans)])
    xlabel(hScan,'Scan Number');                          %create labels for Scan Trace graph
    ylabel(hScan,'Intensity'  );
    title (hScan,'Scan Trace' );
%create the Total Trace graph
    mn=find(delays<=handles.settings.trmin,1,'last');
    mx=find(delays>=handles.settings.trmax,1,'first');
    plot(delays(mn:mx),handles.Data.Trace(mn:mx),'Parent',hTrace);%plot graph
    axis  (hTrace,'tight');
    xlabel(hTrace,'Time Delay');                          %create labels for Total Trace graph
    ylabel(hTrace,'Intensity');
    title (hTrace,'Total Trace');
%create Spectrum Graph
    titl = 'Spectrum'; ylab='Intensity';leg='';
    if ~abel, rad  = sum(rInt,1); end                     %if not abel, don't use the rsin0 value
    if bon
        rad = permute(beta,[1,3,2]); titl = 'Anisotropy Fit'; ylab='\beta';
        rad(:,1) = rad(:,1).*rad(:,2)/max(max(rad(:,1)));
        leg = {'\beta_0','\beta_2','\beta_4'};
    end
    plot(hSpec,rad);    xlim(hSpec,[0,w])
    title (hSpec,titl); ylabel(hSpec,ylab); xlabel(hSpec,'Radius (pixels)');%create labels for Spectrum graph
    if ~isempty(leg),legend(hSpec,leg,'Location','NEO'),end
%create image graph
    axes(hIm)
    if get(handles.polimage,'Value'),dat=rInt;
    elseif get(handles.quartimage,'Value'),dat=data(w+1:2*w,w+1:2*w,:);
    else dat=data;
    end
    dat(isnan(dat))=0;
    imshow(dat,scale,'Parent',hIm),axis image off         %plot image
%     surf(); view(0,90),shading interp,axis image off    %plot image

%update UI
    if ~isempty(textdisplay),set(handles.displaytxt,'String',textdisplay),end %set text
    set([handles.ProcessData fh],'Colormap',mycmap);          %renew colormap on all figures

%deal with exporting figures
    if setting                                            %if exporting a figure
        switch setting
            case 1; ud = rD; case 2; ud = dat; otherwise, ud=1; %get 'UserData' to give to exporting figure (so user
can save data file) (ud=1 retrieves data from graph)
        end
        if setting==2, colorbar('Location','North','Position',[0.3 0.92 0.4 0.03]);end  %horazontal colorbar
        pubfig(fh, [nm name], path,  sz, ud)               %go to external pubfig function
        if setting==1, rowexp(fh,rD,tx,ty,a); end         %make pretty row of images plot
    end
    guidata(handles.xx,handles)
catch   %this is important because if it crashes unchecked, then the scroll image goes funny (crashes the whole
program)
    err = lasterror; errordlg(err.message)
    guidata(handles.xx,handles)
    set(handles.ProcessData,'Colormap',mycmap)                %keep the nice colormap even if it's crashed
%     rethrow(err)
    disp('error on lines - '),err.stack.line,err.message
end

%% Make the Pretty 'Save Row of Images' plot
function ph=rowexp(fh,rD,tx,ty,a)
figure(fh), axis off                                  %hide the default axis (we won't use it)
mycmap= get(fh,'Colormap');  mycmap(end,:) = [1 1 1];   %make the maximum values white (hide empty pictures)
row   = ceil(length(a)/10);  left = rem(length(a),10);  %work out how many rows to plot the images in
rl    = 10; FS=15; if row==1,rl   = length(a); FS=13; end %work out how many pics are in each row (default is 10)
ty    = ty(1:rl)+ty(1)*1.1;  tx   = tx(1,1:rl)-tx(1,1)*0.3;%work out positioning of text labels (adjusted because of
new font size)
data  = ones(size(rD,1),size(rD,2)*rl,row)+0.01;        %allocate space for the images
aa    = cell(rl,row);                                   %allocate space for the text labels
```

```
rD(rD>1)=1;
for i = 1:row-(left~=0)                                  %for each complete row:
    d          = rD(:,:,((i-1)*10+1):i*10);              %get all the images for that row
    data(:,:,i)= d(:,:);                                 %turn them into one long image
    aa(:,i)    = a(((i-1)*10+1):i*10);                   %get all the text labels for that row
end
if left                                                  %if there is an incomplete row
    if isempty(i),i = 0; end                             %if the whole figure is 1 incomplete row
    d = rD(:,:,(i*rl+1):end);                            %get all the images for the row
    data(:,1:size(rD,2)*left,row) = d(:,:);              %turn them into one long image
    aa(1:left,row) = a ((i*rl+1):end);                   %get all the text labels for the row
end
for i=1:row                                              %plot each row:
    axes('Units','Normalized','Position',[0 (row-i)/row-0.01 1 1/(row*1.2)]) %make a new axis in the correct position
    ph = imshow(data(:,:,i),[0 1.01]);                   %display the row in the axis
    text(tx(1,1:rl),ty,aa(:,i),'FontSize',FS)            %put the text labels above the row
end
set(fh,'Colormap',mycmap)                                %turn empty pics white (the max section of the colormap set to
[1 1 1] = white)
set(gca,'FontSize',FS)                                    %this doesn't change anything, but allows the toolbar 'change
font size' buttons to know the font size

%% Export Data
function savedata(handles,type)
global data OldPath NewPath                                          %#ok<NUSED>
%type=1: export, type=2: save as
%Get Data
y   = handles.settings.piccentre(1);                     %get y value of centre
x   = handles.settings.piccentre(2);                     %get x value of centre
w   = handles.settings.picwidth;                         %get width of images
sym = get(handles.symmetrize,'Value');
abel= get(handles.AbelTranform,'Value')||get(handles.halfimage,'Value');
rD  = picproces(handles.Data.rData,[x y w],handles.settings.shotspp,sym,abel,0);
bD  = picproces(handles.Data.bData,[x y w],handles.settings.shotsb,sym,abel,0);
cD  = picproces(handles.Data.cData,[x y w],handles.settings.shotsc,sym,abel,0);
delays = handles.settings.delays;
err=''; fa=''; %#ok<NASGU>
%create message box
if ~abel && type==1 %if no abel transformation
    warndlg('The dataset you are saving does NOT have Abel Transformation applied.')
    uiwait, fa='_nA'; %wait till they say OK
end
[OldPath,Oname,ext] = fileparts(handles.settings.settfile);
[Norm NewPath]      = getpath;
if strcmp(handles.settings.ftype,'Image'), name=['\' Oname];%if image file
else name=['\' Oname(1:end-8)]; end %if scan file

%get suggested filename/location
if type==1,
    cfile = [NewPath name fa '_processed.mat'];
    filetypes = {'*.mat','Full Save (*.mat)';'*.dat','Quick Save - Helmut''s Format (*.dat)'};
    title='Save Processed Data as';
else
    cfile=[NewPath name '.mat'];
    filetypes = {'*settings.mat','Scan File (*settings.mat)';'*.mat','Image File (*.mat)'};
    title='Save Raw Data as';
end
[pfile, ppath, filter] = uiputfile(filetypes,title,cfile);%open "save file" dialog
s           =   [length(pfile)   (strfind(pfile,'.mat'))-1   (strfind(pfile,'.dat'))-1   (strfind(pfile,'processed'))-2
(strfind(pfile,'settings'))-2]; %check to see if file has a .mat or processed in it
pfile= ['\' pfile(1:min(s))];
if type==2, NewPath=ppath; end
if pfile~=0                                      %#ok<BDSCI> %if they choose a file
    %copy data files to "Good Data" Folder
    if ~strcmp(OldPath, NewPath) || type==2
        if type==1; wbt='Backing Up Data...'; else, wbt='Saving Data'; end
        h = waitbar(0.01,wbt);
        settfile = [NewPath pfile 'settings' ext];
```

```
        scannum  = handles.settings.scans(end); piccentre= handles.settings.piccentre;  %#ok<NASGU>
        shotspp  = handles.settings.shotspp;    shotsc   = handles.settings.shotsc;     %#ok<NASGU>
        shotsb   = handles.settings.shotsb;                                             %#ok<NASGU>
        Isize    = handles.settings.size;       Dataa    = rD; Datab = bD; Datac = cD;  %#ok<NASGU>
        c.w      = handles.settings.picwidth;   c.lr     = c.w/2; c.ud = c.w/2;
        if (type==2 && filter==2)||(type==1 && size(rD,3)>1)
            try timestamp= handles.settings.timestamp;catch timestamp='';end   %#ok<NASGU>
            save([NewPath pfile ext],'data','c','timestamp')
        else
            save(settfile,'delays','shotspp','shotsb','shotsc','scannum','Isize','piccentre')
            save([NewPath pfile '_-1' ext],'Dataa','Datab','Datac')
        end
        for i=handles.settings.scans
            waitbar(i/scannum)
            OldFile = [name  '-'  num2str(i) ext];
            NewFile = [pfile '_-' num2str(i) ext];
            try,copyfile([OldPath OldFile],[NewPath NewFile]),end
        end
        close(h)
    end
    if size(rD,3)==1,delays=1:2;rD(:,:,2)=rD; bD(:,:,2)=bD; cD(:,:,2)=cD; end %if there's only one image
    if type==1
        if filter==1; %.mat file
            %make sure filename is good
            pfile=[pfile 'processed.mat']; %add processed.mat to end of name (removing any previous .mat or processed
first)
            %make variables to save
            mx          = (max(max(max(rD(:,1:round(w*get(handles.Scl,'Value')),:)))));
            Isize       = handles.settings.picwidth;
            imdata      = rD/mx;                                        %#ok<NASGU>
            scantrace   = [handles.Data.intenx;handles.Data.inteny];    %#ok<NASGU>
            scanrange   = handles.settings.scans;                       %#ok<NASGU>
            %integrate image and fit beta (go to outside function)
            if get(handles.pump,'Value'),rD=rD-bD;end
            if get(handles.probe,'Value'),rD=rD-cD;end
            [intdata            beta            rad]=IntegrateImage(rD,1,Isize,180,5,0,0,'');            %#ok<NASGU>
%note,IntegrateImage(data,ftype,r,a,av,betaon,errson,txt); r and a control polar images' size, betaon{0|1|2} decides
wether to fit beta, and txt adds txt to waitbar
            %save data
            if any(any(any(intdata))) %if it worked
                prfile=fullfile(ppath,pfile);
                save(prfile,'delays','intdata','beta','rad','imdata','Isize','scantrace','scanrange')
            else err='No file Saved this time.';
            end
        else %.dat file
            pfile=[pfile 'processed.dat']; %add processed.dat to end of name
            [rData          rBeta           rRad]=IntegrateImage(rD,1,handles.settings.picwidth,180,5,0,0,'Pump-Probe
');%note,IntegrateImage(data,r,a); r and a control polar images' size
            [bData       bBeta       bRad]=IntegrateImage(bD,1,handles.settings.picwidth,180,5,0,0,'Pump       alone
');%note,IntegrateImage(data,r,a); r and a control polar images' size
            [cData       cBeta       cRad]=IntegrateImage(cD,1,handles.settings.picwidth,180,5,0,0,'Probe       alone
');%note,IntegrateImage(data,r,a); r and a control polar images' size
            if  any(any(any(rData)))
                rRad=rRad';
                bRad=bRad';
                cRad=cRad';
                fid = fopen(fullfile(ppath,pfile),'wt');
                fprintf(fid, '# TRPES-data, exported into Helmut''s format from Process_Data \n#\n');
                delaysN=length(delays);
                fprintf(fid,     'Kommentar     =     pumPOnly_substracted=%d;     probEOnly_substracted=%d;     X-
scale=%s;\n',get(handles.pump,'Value'),get(handles.pump,'Value'),'et');
                fprintf(fid, 'schritte = %d \n', delaysN); %or "n1+n2"
                fprintf(fid, 'lschritte = %d \n', delaysN);
                fprintf(fid, 'Kanalanzahl = %d \n', handles.settings.picwidth);
                fprintf(fid, 'data_type = TRPES \n');
                fprintf(fid, 'Wellenlaengen = ');
                fprintf(fid, '%6.1f ', 1:handles.settings.picwidth);
                fprintf(fid, '\n');
```

```
                fprintf(fid, 'XMode = SingleX \n');
                if  any(any(any(bData)))
                    fprintf(fid, '*\n\n Pump Alone Data = \n\n');
                    for i=1:delaysN
                        %Every line starts with delay in femtosec which is followed by corresponding PES
                        fprintf(fid, '%6.4f ', delays(i));
                        fprintf(fid, '%6.4f ', bRad(i,1:handles.settings.picwidth));
                        fprintf(fid, '\n');
                    end;
                end
                if  any(any(any(cData)))
                    fprintf(fid, '*\n\n Probe Alone Data = \n\n');
                    for i=1:delaysN
                        %Every line starts with delay in femtosec which is followed by corresponding PES
                        fprintf(fid, '%6.4f ', delays(i));
                        fprintf(fid, '%6.4f ', cRad(i,1:handles.settings.picwidth));
                        fprintf(fid, '\n');
                    end;
                end
                fprintf(fid, 'Data = \n#\n');
                for i=1:delaysN
                    %Every line starts with delay in femtosec which is followed by corresponding PES
                    fprintf(fid, '%6.4f ', delays(i));
                    fprintf(fid, '%6.4f ', rRad(i,1:handles.settings.picwidth));
                    fprintf(fid, '\n');
                end;
                fclose(fid);
            else err='No file Saved this time.';%if it didn't work
            end
        end
        if isempty(err)
            answer=questdlg(['Saved as: ' pfile sprintf('\nDo you want to open this file in Analyse Data?')]);
%display message box
            if strcmp(answer,'Yes'),Analyse_Data_1_0('file',fullfile(ppath,pfile)),end  %Go to PlotEv8.m program
        end
    end
    if isempty(err),txt = ['Saved to: ' fullfile(ppath,pfile)];
    else,txt = err; end %tell user about error
    set(handles.displaytxt,'String',txt);
end
guidata(handles.scans,handles)

%% Scans Editbox
function scans_Callback(hObject, eventdata, handles)
h=eval(get(hObject,'String'));
handles.settings.scans=h;
guidata(hObject,handles)
loaddata(handles,handles.settings.settfile)             %go to 'Load Function'


%% Centre Section

% ARROW BUTTONS

% [<] Button
function pcentreLEFT_Callback(hObject, eventdata, handles), arrow(hObject,handles,2,+1)
% [>] Button
function pcentreRIGHT_Callback(hObject, eventdata, handles),arrow(hObject,handles,2,-1)
% [/\] Button
function pcentreUP_Callback(hObject, eventdata, handles),   arrow(hObject,handles,1,+1)
% [\/] Button
function pcentreDOWN_Callback(hObject, eventdata, handles), arrow(hObject,handles,1,-1)
%arrow function
function arrow(hObject,handles,cn,ng)
%cn=1 (up-down) OR 2 (left-right); ng=1 (left-up) or -1 (right-down)
set(hObject,'Enable','off')
handles.settings.piccentre(cn)=handles.settings.piccentre(cn)+ng*1; %change centre point by 1 pixel
guidata(hObject,handles)   %save values
```

```
picwidthedit(guidata(gcbo));
set(hObject,'Enable','on')

% EDIT BOXES
function xx_Callback(hObject, eventdata, handles)
xyedit(hObject,handles,2)

function yy_Callback(hObject, eventdata, handles)
xyedit(hObject,handles,1)

%XY Edit fn: Gets the XY value, checks it's good, and saves it
function xyedit(hObject,handles,cn)
value = str2double(get(hObject, 'String'));             %get the xy value
W     = handles.settings.size;
if isempty(get(hObject, 'String'))
    handles.settings.piccentre(cn)=W(cn)-handles.settings.picwidth;%go to the maximum possible value
elseif isnan(value)||value>W(cn)-5||value<5             %display error message
    errordlg(['Input must be a number between 5 and ' num2str(W(cn)-5)],'Error')
else, handles.settings.piccentre(cn)=value;             %save value
end
guidata(hObject,handles)                                %Save handles
picwidthedit(guidata(gcbo));

%Pic Width Edit fn: For anything in the 'Crop&Centre' box, check numbers and update the UI
function w=picwidthedit(handles)
c  = handles.settings.piccentre;
W  = handles.settings.size;                             %picture width
w  = round(min([c(1) c(2) W(1)-c(1) W(2)-c(2) handles.settings.picwidth]));     %check/calculate new width
set(handles.xx,'String',c(2)),set(handles.yy,'String',c(1)),set(handles.picwidth,'String',w*2)
handles.settings.picwidth = w; handles.settings.piccentre = c; %save values
guidata(gcbo,handles),picdisp(guidata(gcbo),0,'')       %go to 'Picture Function'

% picwidth Editbox
function picwidth_Callback(hObject, eventdata, handles)
%get data
picwidth = str2double(get(hObject, 'String'));          %get the 'Width' value
wd=floor(picwidth/2);                                   %work with 1/2 the width (rounded down)
x=handles.settings.piccentre(2);                        %piccenter values (x,y)
y=handles.settings.piccentre(1);
w=handles.settings.picwidth;                            %previous width (in case user enters faulty width)
W=handles.settings.size;                                %width of original data
[m I] = min([x y W(2)-x W(1)-y]);                       %shortest distance from the centre point to the edge
if isempty(get(hObject, 'String'))
    handles.settings.picwidth=m;
%error messages:
elseif isnan(picwidth)                                   %if contains text or charicters
    set(hObject, 'String', w*2)                          %go back to previous value
    errordlg('Input must be a number','Error'),uiwait    %display error message
elseif any(picwidth > W)                                 %if it's too big
    set(hObject, 'String', w*2)                          %go back to previous value
    errordlg(['Input cannot be above maximum picture width (' num2str(min(W)) ')'],'Error'),uiwait%display error
message
elseif picwidth < 10                                     %if it's too small
    set(hObject, 'String', w*2)                          %go back to previous value
    errordlg('Picture width must be greater than 9','Error'),uiwait%display error message
%save data:
elseif wd > m                                          %if image too large for centre point to work
    button=questdlg('Making the picture this large will move the centre point. Do you want to continue?');
    if strcmp(button,'No'),set(hObject, 'String', w*2)  %if user presses 'No', go back to previous value
    else                                                %if user presses 'Yes', calculate new piccentre:
        for i=1:2                                       %for both x and y
            switch I                                    %change smallest centrepoint to accommidate larger width
                case 1, x=wd;
                case 2, y=wd;
                case 3, x=W(2)-wd;
                case 4, y=W(1)-wd;
            end
            [m I] = min([x y W(2)-x W(1)-y]);           %find next smallest centrepoint
```

```
            if wd <= m, break, end                      %if it's ok, carry on. Otherwise change it too.
        end
        handles.settings.piccentre= [y x];              %Save the new piccentre
        handles.settings.picwidth = wd;                 %Save the new picwidth
    end
else                                                    %If there are no problems
    handles.settings.picwidth=wd;                       %Save the new picwidth
end
guidata(hObject,handles)                                %Save handles
picwidthedit(guidata(gcbo));

%% Image Section
function  delays_Callback(hObject,  eventdata,  handles),set(handles.imsel,'Value',0),picdisp(handles,0,'')%  Delays
Listbox
function pump_Callback(hObject, eventdata, handles),       picdisp(handles,0,'')% Pump Tickbox
function probe_Callback(hObject, eventdata, handles),      picdisp(handles,0,'')% Pump Tickbox
function AbelTranform_Callback(hObject, eventdata, handles),picdisp(handles,0,'')% Abel Tranform Tickbox
function symmetrize_Callback(hObject, eventdata, handles), picdisp(handles,0,'')% Symmetrize Tickbox
function normimage_Callback(hObject, eventdata, handles),  picdisp(handles,0,'')% Normal Tickbox
function quartimage_Callback(hObject, eventdata, handles), picdisp(handles,0,'')% 1/4 Tickbox
function halfimage_Callback(hObject, eventdata, handles),  picdisp(handles,0,'')% 1/2 Abel Tickbox
function polimage_Callback(hObject, eventdata, handles),   picdisp(handles,0,'')% Polar Tickbox
function logimage_Callback(hObject, eventdata, handles),   picdisp(handles,0,'')% log scale Tickbox
function rawimages_Callback(hObject, eventdata, handles),  picdisp(handles,0,'')% Raw Images Tickbox
function hascale_Callback(hObject, eventdata, handles),    picdisp(handles,0,'')% half abel slider
function imsel_Callback(hObject, eventdata, handles) %'Select All' Tickbox
persistent vals                             %save old set of values
if                              get(hObject,'Value'),                           vals=get(handles.delays,'Value');
set(handles.delays,'Value',1:length(handles.settings.delays))%select all
else set(handles.delays,'Value',vals),end   %deselect all (put back to how it was before)
picdisp(handles,0,'')
function Scl_Callback(hObject, eventdata, handles)%scale slider
set(handles.scltxt,'String',[num2str(get(hObject,'Value')*100) '%'])
picdisp(handles,0,'')
%% Side Graph Section
function betaon_Callback(hObject, eventdata, handles),     picdisp(handles,0,'')

% Trace max
function trmax_Callback(hObject, eventdata, handles)
handles.settings.trmax=tredit(hObject,handles,handles.settings.trmax);
guidata(hObject,handles), picdisp(handles,0,'')         %go to 'Picture Function'
% Trace min
function trmin_Callback(hObject, eventdata, handles)
handles.settings.trmin=tredit(hObject,handles,handles.settings.trmin);
guidata(hObject,handles), picdisp(handles,0,'')         %go to 'Picture Function'

function value=tredit(hObject,handles,hValue)
value=str2double(get(hObject,'String')); del=handles.settings.delays;
if isnan(value)||value<min(del)||value>max(del)
    errordlg(['Input must be a number between ' num2str(min(del)) ' and ' num2str(max(del)) ])
    set(hObject,'String',num2str(hValue)); value=hValue;
end

%% File Menu
function FileMenu_Callback(hObject, eventdata, handles)
function                        loadf_Callback(hObject,                     eventdata,                          handles),
loadsettdata(handles,fileparts(handles.settings.settfile),1)%Load New Data Files
function addf_Callback(hObject, eventdata, handles),        adddatafile(handles)%Add New Data File
function savef_Callback(hObject, eventdata, handles),       savedata(handles,1)%Export Data
function saveas_Callback(hObject, eventdata, handles),      savedata(handles,2)%Save data
function gendata_Callback(hObject, eventdata, handles),     ImageGenerator%Go to ImageGenerator.m program
function prismcomp_Callback(hObject, eventdata, handles),   prismcomplength %go to prismcomplength.m program
function ImageAdd_Callback(hObject, eventdata, handles),    addimage(handles,2)
function ImageSubtract_Callback(hObject, eventdata, handles),addimage(handles,3)
function EditDelays_Callback(hObject, eventdata, handles)
answer          =          inputdlg('Please          Carefully          Type          new          Time          Values
in','',length(handles.settings.delays),{char(get(handles.delays,'String'))});
if isempty(answer),return,end               %if user presses 'cancel'
```

```
handles.settings.delays = str2num(answer{:});  %make sure that they are all numbers
if isempty(handles.settings.delays),errordlg('please make sure all the values are valid numbers (no text)'),return,end
set(handles.delays,'String',cellstr(answer)),guidata(hObject,handles),picdisp(handles,0,'')%save values

%% Figures Menu
function FigureMenu_Callback(hObject, eventdata, handles)
function BackShow_Callback(hObject, eventdata, handles)                    %Show Background Images
if strcmpi(get(hObject,'Checked'),'on'),set(hObject,'Checked','off')
else set(hObject,'Checked','on'),end, picdisp(handles,0,'')
function SelectShow_Callback(hObject, eventdata, handles)                  %Show Selected Images
if strcmpi(get(hObject,'Checked'),'on'),set(hObject,'Checked','off')
else set(hObject,'Checked','on'),end, picdisp(handles,0,'')
function CmpEdit_Callback(hObject, eventdata, handles), colormapeditor     %Edit Colormap
function                    CmpSave_Callback(hObject,                      eventdata,                      handles),
cmp=get(handles.ProcessData,'ColorMap');uisave('cmp','MyColormaps/ProcessDefault.mat') %#ok<NASGU>
function                    CmpLoad_Callback(hObject,                      eventdata,                      handles),
uiopen('MyColormaps/ProcessDefault.mat'),set(handles.ProcessData,'ColorMap',cmp)
function RowSave_Callback(hObject, eventdata, handles),  picdisp(handles,1,'')%Save Row of Images
function ImSave_Callback(hObject, eventdata, handles),   picdisp(handles,2,'')%Save Side Images
function SpecSave_Callback(hObject, eventdata, handles), picdisp(handles,3,'')%Save Side Graphs
function ScanSave_Callback(hObject, eventdata, handles), picdisp(handles,4,'')%Save Side Graphs
function TraceSave_Callback(hObject, eventdata, handles),picdisp(handles,5,'')%Save Side Graphs
%% Help Menu
function HelpMenu_Callback(hObject, eventdata, handles)
function AbHelp_Callback(hObject, eventdata, handles),fhelp('About')
function PDHelp_Callback(hObject, eventdata, handles),fhelp('Process Data Help')
function CCHelp_Callback(hObject, eventdata, handles),fhelp('Crop and Centre Help')
function ImHelp_Callback(hObject, eventdata, handles),fhelp('Image Help')
function ExHelp_Callback(hObject, eventdata, handles),fhelp('Exporting Data Help')
function STHelp_Callback(hObject, eventdata, handles),fhelp('Scan Trace Help')

function fhelp(subject)
if strcmpi(subject,'About')
    helptxt=['This software was written by Ruth Livingstone over the course of'...
    ' her PhD. It is designed to open data files created by Camera_Stage.m .'...
    ' These files should be in the form of a ...settings.mat file containing'...
    ' the main variables required, coupled with scan files (...-1.mat, ...-2.mat etc)'...
    ' containing the data from each scan. It will take this data, process it and export it'...
    ' into a form readable by PlotE.mat. It was written using matlab 7.4.0(R2007a)'];
elseif strcmpi(subject,'Process Data Help')
    helptxt='Process Data Help Text Here';
elseif strcmpi(subject,'Crop and Centre Help')
    helptxt='Crop and Centre Help Text Here';
elseif strcmpi(subject,'Image Help')
    helptxt='Image Help Text Here';
elseif strcmpi(subject,'Exporting Data Help')
    helptxt='Exporting Data Help Text Here';
elseif strcmpi(subject,'Scan Trace Help')
    helptxt='Scan Trace Help Text Here';
end
helpdlg(sprintf([subject ':\n\n' helptxt]),subject);

%% Create Functions
function scans_CreateFcn(hObject, eventdata, handles),        createf(hObject,'white')
function xx_CreateFcn(hObject, eventdata, handles),           createf(hObject,'white')
function yy_CreateFcn(hObject, eventdata, handles),           createf(hObject,'white')
function picwidth_CreateFcn(hObject, eventdata, handles),     createf(hObject,'white')
function trmax_CreateFcn(hObject, eventdata, handles),        createf(hObject,'white')
function trmin_CreateFcn(hObject, eventdata, handles),        createf(hObject,'white')
function delays_CreateFcn(hObject, eventdata, handles),       createf(hObject,'white')
function hascale_CreateFcn(hObject, eventdata, handles),createf(hObject,[.9 .9 .9])
function Scl_CreateFcn(hObject, eventdata, handles),          createf(hObject,[.9 .9 .9])
function createf(hObject,colour)
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',colour);
end
```

Appendix.D. Pulse Propagation Calculations

```
function varargout = prismcomplength(varargin)
% Calculates Various optical proparties
%      PRISMCOMPLENGTH, by itself, creates a new PRISMCOMPLENGTH or raises the existing
%      singleton*.
%
%      H = PRISMCOMPLENGTH returns the handle to a new PRISMCOMPLENGTH or the handle to
%      the existing singleton*.
%
%      PRISMCOMPLENGTH('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PRISMCOMPLENGTH.M with the given input arguments.
%
%      PRISMCOMPLENGTH('Property','Value',...) creates a new PRISMCOMPLENGTH or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before prismcomplength_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to prismcomplength_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help prismcomplength

% Last Modified by GUIDE v2.5 18-Jan-2012 18:19:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @prismcomplength_OpeningFcn, ...
                   'gui_OutputFcn',  @prismcomplength_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before prismcomplength is made visible.
function prismcomplength_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for prismcomplength
handles.output = hObject;
%set initial values (feel free to change any of these)
handles.value.wav       =267;           %wavelength
handles.value.tlpd      =80;            %transform limited pulse length
handles.value.apd       =200;           %actual pulse length
handles.value.D         =4;             %distance from tip
handles.value.cb        =0.441;         %Gaussian pulse profile (see pulseshape_Callback for more possible values)
handles.value.length    =35;            %User chosen prism compresser length
handles.value.anglep    =56.3;          %Angle in to the prisms
handles.value.angleo    =0;             %Angle in to the window/lens
handles.value.calculation=1;            %Choose Prism Compresser Calculation
handles.value.ba        =56.3;          %brewster's angle
handles.value.prismangle=56;            %angle into prism
% put these values into user interface
set (handles.wav,       'String', handles.value.wav   )
set (handles.tlpd,      'String', handles.value.tlpd  )
set (handles.apd,       'String', handles.value.apd   )
```

```
set (handles.D,         'String', handles.value.D     )
set (handles.length,    'String', handles.value.length)
set (handles.angle,     'String', handles.value.anglep)
set (handles.prismangle,'String', handles.value.prismangle)
axes(handles.setup), imshow('prismcompsingle.bmp' ) %display diagram of prism compresser
% Update handles structure
guidata(hObject, handles)
% Run Calculation
calculate(handles)

% --- Outputs from this function are returned to the command line.
function varargout = prismcomplength_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
%% Calculation Functions

%MAIN CALCULATE FUNCTION
function calculate(handles)
%define input parameters
w   =handles.value.wav;                    %centre wavelength
tlpd=handles.value.tlpd;                   %transform limited pulse length
apd  =handles.value.apd;                   %actual pulse length
D    =handles.value.D;                     %distance from tip or thickness of optic
cb   =handles.value.cb;                    %pulse shape
ppol =get(handles.ppol,'Value');           %polarisation
glass=get(handles.prismmaterial,'Value');
prismangle=handles.value.prismangle*pi/180; %change prism angle to radians
configuration=get(handles.confuguration,'Value');%double or single pass
%calculate and define parameters used in calculations
wav=w/1000;                                %wavelength in micrometers
bw=w^2*cb/(300*tlpd);                      %bandwidth
fw=300*bw/(w^2);                           %frequency width
GDDp=1/(4*log(2))*sqrt((cb*apd/fw)^2-(cb/fw)^4);%pulse GDD
ab=absorbance(wav,glass,handles.transcurve); %transmission
[n dn ddn]=sellmeier(wav,glass);

axes(handles.graph)
if handles.value.calculation==1;               %prism compresser
    beta=-2*dn*bw/1000;
    %calculate pulse length at user defined prism compresser length
    [handles.value.pd length]=prismcompress(handles.value.length,w,n,dn,ddn,beta,D,GDDp,tlpd,configuration);
    x=linspace(0,length*3,100);                %make stuff for graph
    y=prismcompress(x,w,n,dn,ddn,beta,D,GDDp,tlpd,configuration);%calculate graph data
    d=D*sin(prismangle)/sin(pi-prismangle/2);  %distance light travels inside prism
    angle=handles.value.anglep*pi/180;         %angle hitting first prism
    angleback=acos(sqrt(1-(1/n*sin(angle))^2))-prismangle;%angle hitting back face of prism
    if configuration==1;r=5;d=d*4;else r=4;d=d*2;end        %choose which reflection calc to use
    %Display answers
    string=['\nOptimum Compressor Length is: ' num2str(length,'%-5.1f') ' cm'];
    xl='Compresser Length (cm)';
elseif handles.value.calculation==2;           %window/lens
    angle=handles.value.angleo*pi/180;         %angle hitting optic
    angleback=acos(sqrt(1-(1/n*sin(angle))^2)); %angle hitting back face of optic
    D=D/acos(angleback);                       %change optic thickness to reflect angle
    handles.value.pd=optic(D,w,tlpd,glass,configuration);%calculate pulse duration at optic thickness
    x=linspace(w-200,w+200,1000);x(x<170)=170;x(x>3000)=3000;%make stuff for graph
    y=optic(D,x,tlpd,glass,configuration);     %main window/lens calculation
    y(y>tlpd*20)=tlpd*20;                      %get rid of large numbers
    [t l]=walkoff(n,dn,w,D,configuration);     %Pulse Delay Calculation
    if configuration==1;r=4;d=D*2;else r=3;d=D;end%choose which reflection calc to use
    %Display answers
    if configuration==2; s=' ';else,s='s ';end
    string=['Pulse is lengthened by optic' s 'by: ' num2str(handles.value.pd-tlpd,'%-5.3f')...
        ' fs\n\nBeam has been delayed by: ' num2str(t,'%-5.2f') ' ps (' num2str(l,'%-5.2f') ' mm)'];
    set(handles.lengthtxt,'String',sprintf(['\nPulse Duration is now:  ' num2str(handles.value.pd,'%-5.1f')...
         ' fs\nDouble Pass Path Length Offset: ' num2str(l*500,'%-5.0f') ' um']))
    xl='Centre Wavelength (nm)';
end
%Reflection Calculation
```

```
al=1-10^(-ab*d);                                %absorptive loss NOT WORKING PROPERLY
[Rsf Rpf]=reflection(1,n,angle);                %Reflective Loss on front face
[Rsb Rpb]=reflection(n,1,angleback);            %Reflective Loss on back face
if ppol,R=[Rpf Rpb];                            %p-polarised light
else,R=[Rsf Rsb];end                            %s-polarised light
R(3)=R(1)+(1-R(1))*R(2);                        %total reflective loss per optic
R(4)=R(3)+(1-R(3))*R(3);                        %2 optics
R(5)=R(4)+(1-R(4))*R(4);                        %4 optics
Rt=R(r)+(1-R(r)/2)*(al);                        %Choose which value to use and facter in absorbtion
if Rt>1,Rt=1; end

%save answers
bwcm    = (10000000/(w-bw)-10000000/(w+bw))/2;
bweV    = (1240/(w-bw)-1240/(w+bw))/2;
handles.value.bw     = bw;
handles.value.ba     = atan(n)*180/pi;
guidata(handles.n,handles)
%display answers
set(handles.disptxt,'String',sprintf(string))
%set(handles.length,'String',num2str(handles.value.length,'%-5.2f'))
set(handles.pd,  'String',num2str(handles.value.pd,'%-5.0f'))
set(handles.tlpd,'String',num2str(tlpd,'%-5.0f'))
set(handles.bwnm,'String',num2str(bw,3))
set(handles.bwcm,'String',num2str(bwcm,'%-5.0f'))
set(handles.bweV,'String',num2str(bweV,2))
set(handles.n,   'String',num2str(n,'%-5.2f'))
set(handles.ba,  'String',num2str(handles.value.ba,'%-5.1f'))
set(handles.Rf,  'String',[num2str(R(1)*100,'%-5.2f') '%'])
set(handles.Rb,  'String',[num2str(R(2)*100,'%-5.2f') '%'])
set(handles.ab,  'String',[num2str(  al*100,'%-5.2f') '%'])
set(handles.Rt,  'String',[num2str(  Rt*100,'%-5.2f') '%'])
plot(handles.graph,x,y), axis tight
xlabel(xl);ylabel('Pulse Length (fs)');
% x=200:2800;y=optic(D,x,tlpd,glass,configuration);figure,plot(x,y),pubfig %(used to make plot for my thesis)

%OTHER CALCULATIONS (called from within calculate function)

%'absorbance' estimates the attenuation coefficient of the material at
%that wavelength, and displays a tranmission curve
function ab=absorbance(wav,glass,graph)
%transmission curves:
FS=[170 200 400 500 700 900 1000 1300 1385 1500 1600 2000 2050 2150 2200 2300 2380 2500 2720 2800 2900 3000;...
0.63 0.96 0.999 0.999 0.999 0.999 0.999 0.999 0.88 0.99 0.999 0.999 0.99 0.95 0.58 0.88 0.95 0.79 0 0 0.295 0.67];
CF=[110 150 200 270 350 500 1000 3000 5000 6000 7000 8000 9000 10000;...
0.1 0.6 0.8 0.9 0.97 0.97 0.99 0.99 0.99 0.98 0.97 0.88 0.59 0.19];
SF10=[110 360 370 380 390 400 420 440 460 480 500 550 600 650 700 800 900 1000 1200 1400 1600 1800 2000 2200 2400;...
0 0 0.09 0.41 0.67 0.81 0.931 0.963 0.975 0.982 0.987 0.994 0.995 0.993 0.994 0.998 0.998 0.998 0.998 0.994 0.993
0.985 0.977 0.947 0.929];
BK7=[110 280 290 300 310 320 330 340 350 360 370 380 390 400 420 440 460 480 500 550 600 650 700 800 900 1000 1200
1400 1600 1800 2000 2200 2400;...
0 0 0.08 0.31 0.58 0.77 0.88 0.94 0.968 0.984 0.991 0.991 0.996 0.997 0.996 0.995 0.995 0.996 0.996 0.998 0.997 0.997
0.998 0.998 0.997 0.996 0.995 0.982 0.991 0.98 0.961 0.89 0.85];
LiF=[118 140 200 310 500 1000 3000 5000 6000 7000;...
0.45 0.8 0.9 0.96 0.98 0.97 0.97 0.88 0.65 0.14];
switch glass            %choose material
    case 1, abb=FS;     %Fused Silica
    case 2, abb=CF;     %Calcium Flouride
    case 3, abb=SF10;   %SF10
    case 4, abb=BK7;    %BK7 glass
    case 5, abb=LiF;    %Lithium Flouride
end
axes(graph) %make graph
x=170:2400; y=pchip(abb(1,:),abb(2,:),x);%calculate stuff for graph
semilogx(x,y*100)%,hold on,semilogx(abb(1,:),abb(2,:)*100,'o'),hold off%plot graph
xlabel('Wavelength (nm)'),ylabel('Transmittance (%)') %label axes
axis([170 2400 0 100]),set(gca,'XTick',[200 350 600 1000 2000])%scale axes
T=pchip(abb(1,:),abb(2,:),wav*1000);    %calculate transmission at that particular wavelength
ab=-log10(T)/10;                        %attenuation coefficient
```

```
%'sellmeir' calculates refractive index as a function af wavelength and it's
%derivatives using sellmeir's equations and co-efficients for different materials
function [n dn ddn]=sellmeier(wav,glass)
cFS  =[0.69616630 0.004679148 0.407942600 0.013512063 0.89747940 97.93400254];  %Sellmeier's cooefficients for Fused
Silica
cCF  =[0.61881400 0.002759866 0.419893700 0.010612510 3.42629900 1068.123000];  %Sellmeier's cooefficients for Calcium
Flouride
cSF10=[1.62153902 0.012224146 0.256287842 0.059573678 1.64447552 147.4687930];  %Sellmeier's cooefficients for SF10
cBK7 =[1.03961212 0.00600069867 0.231792344 0.0200179144 1.0104694 103.560653]; %Sellmeier's cooefficients for BK7
cLiF =[0.933385 0.0715768 2.91928 29.3086 2.73548 29.4083]; %Sellmeier's cooefficients for LiF !!NOT SURE IF CORRECT!!
switch glass            %choose material
    case 1, c=cFS;      %Fused Silica
    case 2, c=cCF;      %Calcium Flouride
    case 3, c=cSF10;    %SF10
    case 4, c=cBK7;     %BK7 glass
    case 5, c=cLiF;     %Lithium Flouride
end
n  =sqrt(1+c(1)*wav.^2./(wav.^2-c(2))+c(3)*wav.^2./(wav.^2-c(4))+c(5)*wav.^2./(wav.^2-c(6)));%refractive index
dn =1./(2.*n).*(- 2.*c(1).*c(2)^2*wav./((wav.^2-c(2)).^2) - ...                        %derivative of refractive index
with respect to wavelength
        2*c(3)*c(4)^2*wav./((wav.^2-c(4)).^2) - 2*c(5)*c(6)^2*wav./((wav.^2-c(6)).^2));
ddn=((-(2*c(1)*wav.^3)./(wav.^2-c(2)).^2 + 2*c(1)*wav./(wav.^2-c(2))...                          %second derivative of
refractive index with respect to wavelength
      -(2*c(3)*wav.^3)./(wav.^2-c(4)).^2 + 2*c(3)*wav./(wav.^2-c(4))...
      -(2*c(5)*wav.^3)./(wav.^2-c(6)).^2 + 2*c(5)*wav./(wav.^2-c(6))).^2)./...
     (4*(1 + (c(1)*wav.^2)./(wav.^2-c(2)) + (c(3)*wav.^2)./(wav.^2-c(4)) + (c(5)*wav.^2)./(wav.^2-c(6))).^3/2)+...
    ((8*c(1)*wav.^4)./(wav.^2-c(2)).^3 - (10*c(1)*wav.^2)./(wav.^2-c(2)).^2 + 2*c(1)./(wav.^2-c(2))+...
     (8*c(3)*wav.^4)./(wav.^2-c(4)).^3 - (10*c(3)*wav.^2)./(wav.^2-c(4)).^2 + 2*c(3)./(wav.^2-c(4))+...
     (8*c(5)*wav.^4)./(wav.^2-c(6)).^3 - (10*c(5)*wav.^2)./(wav.^2-c(6)).^2 + 2*c(5)./(wav.^2-c(6))).*...
     (2*sqrt(1+(c(1)*wav.^2)./(wav.^2-c(2)) + (c(3)*wav.^2)./(wav.^2-c(4)) + (c(5)*wav.^2)./(wav.^2-c(6))));
n=real(n)+imag(n);

%prism compresser calculation
function [pulseduration optimumlength]=prismcompress(prismlength,wav,n,dn,ddn,beta,D,GDDp,tlpd,config)
    %Calculate the pulse dispertion going through the prisms
GDDpr=(wav^3)/(2*pi*(300)^2)*(40*prismlength/config*((ddn+(2*n-1/(n)^3)*(dn)^2)*sin(beta)-
2*(dn)^2*cos(beta))+4*ddn*D);
    %calculate the pulse duration in fs after going through the prisms
pulseduration=sqrt(tlpd^4+(16*(log(2))^2*(GDDpr+GDDp).^2))/tlpd;
    %calculate the optimum compresser length
optimumlength=-(GDDp*2*pi*(300)^2/wav^3+4*ddn*D)/(40*((ddn+(2*n-1/(n)^3)*(dn)^2)*sin(beta)-
2*(dn)^2*cos(beta)))*config;

%window/lens pulse duration calculation
function pulseduration=optic(length,wav,pd0,glass,configuration)
wav=wav/1000; %nm to um
length=length*1000*2/configuration;     %mm to um (with single/double pass compensation)
%[n dn ddn]=ddncalc(wav,c,BBO);          %work out how ddn changes with wavelength
[n dn ddn]=sellmeier(wav,glass);
a=((wav).^3)/(4*pi*(0.29979)^2).*ddn;
pulseduration=pd0*sqrt(1+(8*a*length*log(2)/pd0^2).^2);
pulseduration=real(pulseduration);

%Pulse Delay Calculation
function [t l]=walkoff(n,dn,w,D,configuration)
w=w*0.000000001; D=D*0.001*2/configuration; %change to SI units
v=300000000/(n-w*dn);                       %calculate group velocity inside lens or window in m/s
t=(D/v-D/300000000)*1e12;                   %calculate additional time to travel through lens/window in ps
l=t*0.3;                                    %calculate equivilent delay length in mm (length=time*c/1e12(ps)/1e-3(mm))

%Reflective loss Calculation
function [Rs Rp]=reflection(n1,n2,angle)
Rs=((+n1*cos(angle)-n2*sqrt((1-(n1/n2*sin(angle))^2)))/(n1*cos(angle)+n2*sqrt((1-(n1/n2*sin(angle))^2))))^2;
Rp=((-n2*cos(angle)+n1*sqrt((1-(n1/n2*sin(angle))^2)))/(n2*cos(angle)+n1*sqrt((1-(n1/n2*sin(angle))^2))))^2;
if ~isreal(Rs), Rs=1;end, if ~isreal(Rp), Rp=1;end %if get imaginary no.s, set to 100% reflectance

%MISCELANIOUS FUNCTIONS
```

```
%Change optical setup diagram (note - all diagrams named below need to be saved into the same folder as the program)
function setupimage(handles)                %setupimage is called from calculation_Callback
axes(handles.setup)                         %draw to setup axis
if get(handles.calculation,'Value')==1      %if prism compresser
    if get(handles.confuguration,'Value')==2;imshow('prismcompsingle.bmp');%if single pass prism compresser
    else imshow('prismcompdouble.bmp');end  %if double pass prism compresser
else                                        %if window/lens
    if get(handles.confuguration,'Value')==2;imshow('opticsingle.bmp');%if single pass window/lens
    else imshow('opticdouble.bmp');end      %if double pass window/lens
end
%% Inputs
%Calculation Selector {Prism Compresser,Window/Lens}
function calculation_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function calculation_Callback(hObject, eventdata, handles)
value=get(hObject,'Value');
axes(handles.setup)
if value==1,D='Distance from Prism tip (mm):';en='On';vis='Off';a=handles.value.anglep;
elseif value==2, D='Thickness of Optic (mm):';en='Off';vis='On';a=handles.value.angleo;
end
set(handles.Dtxt,'String',D)                %change the label for Dtxt Edit box
set(handles.angle,'String',a)               %change the 'Angle In' Edit box to the correct value
set([handles.prismangle handles.apd],'Enable',en)%Enable/Disable controls only applicable to Prism Compresser Calc
set(handles.lengthtxt,'Visible',vis)        %Show/Hide display text for Window/Lens calculation
handles.value.calculation=value;            %Save calculation choice
guidata(hObject,handles)
setupimage(handles)                         %Change optical setup diagram
calculate(handles)                          %Perform Calculation

% Wavelength
function wav_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function wav_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.wav),errordlg('Please enter a positive number')
else,handles.value.wav=value;guidata(hObject,handles);end
calculate(handles)

% Transfor Limited Pulse Duration
function tlpd_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function tlpd_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.tlpd),errordlg('Please enter a positive number')
else,handles.value.tlpd=value;guidata(hObject,handles);end
calculate(handles)

% Actual Pulse Duration
function apd_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function apd_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.apd),errordlg('Please enter a positive number')
else,handles.value.apd=value;guidata(hObject,handles);end
calculate(handles)

% Distance from tip
function D_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function D_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.D),errordlg('Please enter a positive number')
else,handles.value.D=value;guidata(hObject,handles);end
```

```
calculate(handles)

% Prism Angle
function prismangle_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function prismangle_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.prismangle),errordlg('Please enter a positive number')
else,handles.value.prismangle=value;guidata(hObject,handles);end
calculate(handles)

% Prism material - choose {Fused Silica,Calcium Flouride,SF10 etc}
function prismmaterial_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function prismmaterial_Callback(hObject, eventdata, handles)
calculate(handles)

% Pulse Shape - choose {Gaussian,Sech,Lorentzian,Rectangle}
function pulseshape_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function pulseshape_Callback(hObject, eventdata, handles)
switch get(hObject,'Value')
    case 1, handles.value.cb=0.441;   %Gaussian pulse profile
    case 2, handles.value.cb=0.315;   %Sech pulse profile
    case 3, handles.value.cb=0.142;   %Lorentzian pulse profile
    case 4, handles.value.cb=0.443;   %Rectangle pulse profile
end
guidata(hObject,handles)%save answer
calculate(handles)

%Prism Configuration - choose {single pass,double pass}
function confuguration_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function confuguration_Callback(hObject, eventdata, handles)
setupimage(handles)
calculate(handles)

%% Useful Info
%length
function length_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function length_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.length),errordlg('Please enter a positive number')
else,handles.value.length=value;guidata(hObject,handles);end
calculate(handles)

%bandwidth nm
function bwnm_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function bwnm_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.bw),errordlg('Please enter a positive number')
else,handles.value.tlpd=handles.value.wav^2*handles.value.cb/(300*value);guidata(hObject,handles);end
calculate(handles)

%bandwidth cm-1
function bwcm_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function bwcm_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
```

```
if isnan(value)||value<=0,set(hObject,'String',handles.value.bw),errordlg('Please enter a positive number')
else
    value=((1E+7)/(1E+7/handles.value.wav-value)-(1E+7)/(1E+7/handles.value.wav+value))/2;%turn into nm
    handles.value.tlpd=handles.value.wav^2*handles.value.cb/(300*value);%calculate pulse duration
    guidata(hObject,handles);
end
calculate(handles)

function bweV_CreateFcn(hObject, eventdata, handles)
if                      ispc                      &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function bweV_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if isnan(value)||value<=0,set(hObject,'String',handles.value.bw),errordlg('Please enter a positive number')
else
    value=(1240/(1240/handles.value.wav-value)-1240/(1240/handles.value.wav+value))/2;%turn into nm
    handles.value.tlpd=handles.value.wav^2*handles.value.cb/(300*value);%calculate pulse duration
    guidata(hObject,handles);
end
calculate(handles)
%% Reflective Loss
%Angle
function angle_CreateFcn(hObject, eventdata, handles)
if                      ispc                      &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function angle_Callback(hObject, eventdata, handles)
value=str2double(get(hObject,'String'));
if handles.value.calculation==1,a=handles.value.anglep;
else a=handles.value.angleo;end
if isnan(value),set(hObject,'String',a),errordlg('Please enter a number')
else
    if handles.value.calculation==1,handles.value.anglep=value;
    else handles.value.angleo=value;end
    guidata(hObject,handles);
end
calculate(handles)

%polarisation buttons
function spol_Callback(hObject, eventdata, handles),calculate(handles)
function ppol_Callback(hObject, eventdata, handles),calculate(handles)
```

Appendix.E. Integrate Image and Fit Anisotropy

```
function [rInt beta radius]=IntegrateImage(data,ftype,rad,angle,av,betaon,errson,txt)
%Integrate Image takes a cartesian coordinate image and turns it into a
%polar coordinate image. The final image is plotted intensity vs radius vs
%angle. In addition to this, if required, it fits an anisotropy factor
%(beta) to each angular slice, and returns the beta factor along with the
%intensity at that slice.
%data  = symetric input data. Should be a series of centred and cropped images
%            of dimension [W, W, p], where W is picture width and p is # of pics
%ftype = {1,2} first or second order fitting algorithm
%rad   = number of radial slices to take
%angle = number of angular slices to take
%av    = averages over a wider array of angles (i.e. if av=5, each angle point
%            is the average of 5 angles. Makes sure no pixels are missed)
%betaon= {2,1,0} 0=beta fitting disabled, 1=beta fitting enabled, 2=fit beta to already intigrated data.
%errson= {1,0}   0=don't get errors for fits, 1= get errors for fits,
%txt   = text to display on the waitbar: i.e. Integrating and fitting 'txt' images...
%           (if txt='no' then no waitbar is displayed)
%rInt  = polar coordinate images with dimensions: [angle, rad, p]
%beta  = fitted beta parameter and intensity with dimensions: [rad, p, 2]
%radius= Intensity vs Radius vs Delay final dataset with dimensions: [rad, p]

%work out what user wants to do
if isempty(data),rInt=0;beta=0;radius=0;return,end
data(isnan(data))=0;
bet=rad; h=[]; wb=0; wtxt=['Integrating and fitting ' txt 'images...'];
if ~strcmpi(txt,'no'), wb=1; h=waitbar(0,wtxt,'CreateCancelBtn','delete(gcbf)'); end %set up waitbar if required
%get variables
if betaon>1; rad = size(data,2); angle = size(data,1); av = 1;end %set the things that shouldn't be changed for
betaon=2 fitting
crang= 5;%ANGLE TO CROP OUT (NOT USE IN BETA FITS)
W    = size(data,2);   w  = floor(W/2);                %get width of images
p    = size(data,3);   avs= 2*pi/(av*angle);           %get no. of pictures
Tol= 0.000000001;      beta = zeros(rad,p,1+ftype);    %note: beta(:,:,1)=b0; beta(:,:,2)=b2; beta(:,:,3)=b4;
data(isnan(data)) = 0;
if ndims(bet)>2 && all(size(bet)>=[rad p 4]), beta = bet(1:rad,1:p,1:(1+ftype)); end
mx=max(max(max(data))); data=data/mx;                  %normalise data (makes it quicker to fit)
fitcr=[crang:round(angle/2)-crang round(angle/2)+crang:angle-crang];
%make variables for mapping to polar coordinates
an=linspace(0,2*pi,angle); r=linspace(0,w,rad);        %set the angle and radius ranges
[x y] = meshgrid((1:W)-w,(1:W)-w);                       %make x and y matrices to definine the x-y position of each
old pixel
[r a] = meshgrid(r,an);                                     %make a and r matrices to definine the angle and radius
position of each new pixel
for i=1:av
    [xx(:,:,i) yy(:,:,i)] = pol2cart(a+i*avs-av*avs/2,r); %calculte the x-y position of each new pixel
end

%set up beta fitting
xdata=an(fitcr); lb=[-Inf -1 -1]; ub=[Inf 2 2];%define various variables for the beta parameter fitting
% opt.fobfit='a/(4*pi)*(1+b/2*(3*cos(x).^2-1))';
opt.fobfit='a/(4*pi)*(1+b/2*(3*cos(x).^2-1))';
opt.sobfit='a/(4*pi)*(1+b/2*(3*cos(x).^2-1)+c/8*(35*cos(x).^4-30*cos(x).^2+3))';
opt.f = fitoptions('Method','NonlinearLeastSquares','TolFun',Tol,'Lower',lb(1:2),'Upper',ub(1:2));
opt.s = fitoptions('Method','NonlinearLeastSquares','TolFun',Tol,'Lower',lb,'Upper',ub);
opt.options = optimset('Display','off','TolFun',Tol);    %control fitting function
%assign space for data
radius = zeros(rad,p);
err  = zeros(rad,p,1+ftype);
fInt = zeros(angle,rad,p);
int  = zeros(angle,rad,av);
%calculate stuff for each pic
if betaon<2;                                              %if standard (round) image
    rInt = zeros(angle,rad,p);                            %prepare rectangular image space
    data=permute(data,[2 1 3]);                           %turn images right way round
    for pic=1:p                                           %for each picture
```

```
        if ishandle(h),h = waitbar(pic/p);elseif wb,rInt=0;break,end %update the waitbar, or exit if user has closed
waitbar
        for i=1:av,int(:,:,i) = interp2(x,y,data(:,:,pic),xx(:,:,i),yy(:,:,i));end %transform from cartesian to polar
co-ords
        int(isnan(int))=0;
        rInt(:,:,pic) = mean(int,3);                        %average over several different angles
        radius(:,pic) = sum(rInt(:,:,pic).*r.*r.*abs(sin(a)),1);%get an intensity vs radius trace
        if betaon==1                                        %fit beta perameters to each radial slice
            [beta(:,pic,:)                                                                          err(:,pic,:)
fInt(:,:,pic)]=fitbeta(ftype,errson,beta(:,pic,1:1+ftype),xdata,rInt(fitcr,:,pic),lb,ub,opt,an,rad);
        end
    end
    rInt(isnan(rInt))=0;                                    %get rid of non numbers
else                                                        %if fitting pre-processed data
    rInt = data;                                            %get that data
    for pic=1:p                                             %for each pic
        ydata=rInt(fitcr,:,pic);%.*r(fitcr,:);              %crop out the angles you don't want
        if ishandle(h),h = waitbar(pic/p);elseif wb,rInt=0;break,end %update the waitbar, or exit if user has closed
waitbar
        [beta(:,pic,:)                                                                          err(:,pic,:)
fInt(:,:,pic)]=fitbeta(ftype,errson,beta(:,pic,1:1+ftype),xdata,ydata,lb,ub,opt,an,rad);
    end
    nanInt=ones(size(rInt))*nan;                            %return the image showing only the areas thet were fitted
    nanInt(fitcr,:,:)=rInt(fitcr,:,:);                      %i.e. make all the areas not fitted NaN's
    rInt=nanInt;
    nanInt(fitcr,:,:)=fInt(fitcr,:,:);                      %i.e. make all the areas not fitted NaN's
    fInt=nanInt;
end
%put everything in the right place and rescale
%(scaling is required as data is normalised at the start to speed up fitting)
if ftype==1, err(:,:,3)=0; beta(:,:,3)=0;end
if errson,beta(:,:,4:5)=err(:,:,2:3);end                    %if there are errors required, save them at the end of
beta
beta(:,:,1) = beta(:,:,1)*mx;                               %rescale intensity values
radius = radius*mx;                                         %rescale radius
if any(betaon==[1,2]),rInt=cat(3,rInt,fInt)*mx;             %add fits to the end of data
else rInt=rInt*mx; end                                      %rescale rectangular data
if ishandle(h); close(h), end                               %close waitbar

%OTHER FUNCTIONS
%fit Beta (using a starting point)
function [beta err fInt]=fitbeta(ftype,errson,start,xdata,ydata,lb,ub,opt,an,rad)
% work out what type of fitting you are doing
if     ftype==1; ft1=opt.fobfit; ft2=@fbfit; opts=opt.f; lb=lb(1:2); ub=ub(1:2); %first order fitting (b0 and b2 only)
elseif ftype==2; ft1=opt.sobfit; ft2=@sbfit; opts=opt.s; %second order fitting (b0, b2 and b4)
end
%initialise variables
err=zeros(rad,1,1+ftype); beta=zeros(rad,1,1+ftype); fInt=zeros(length(an),rad,1);
%loop for each slice in the picture
if errson %if you want errors
    for rr=1:rad-1
        c = fit(xdata',ydata(:,rr),fittype(ft1,'options',fitoptions(opts,'Startpoint',start(rr,:,:))));
        err(rr,1,:) = diff(confint(c,0.683));   %get the errors (68.3% = 1 standard deviation)
        if ftype==1,beta(rr,1,:)= [c.a c.b];    %get fit peramiters
        else beta(rr,1,:)= [c.a c.b c.c]; end   %get fit peramiters
        fInt(:,rr,1)= feval(c,an');     %get the fit
    end
else
    for rr=1:rad-1,
        [beta(rr,1,:) err(rr,1,1)]=lsqcurvefit(ft2,start(rr,:,:),xdata,ydata(:,rr)',lb,ub,opt.options);
        fInt(:,rr,1) = ft2(beta(rr,1,:),an'); %get the fit
    end
end

%first order beta fitting (fits b0 and b2 as x(1) and x(2))
function y=fbfit(x,xdata)
y=x(1)/(4*pi)*(1+x(2)/2*(3*cos(xdata).^2-1));     %define fitting function
```

```
%second order beta fitting (fits b4 (x(3)) as well as b0 and b2)
function y=sbfit(x,xdata)
y=x(1)/(4*pi)*(1+x(2)/2*(3*cos(xdata).^2-1)+x(3)/8*(35*cos(xdata).^4-30*cos(xdata).^2+3));

%cheating beta fitting (fit only beta=[2 0] or whatever you want to define)
function y=cbfit(x,xdata)
beta=[2 0];
y=x(1)*(1+0.5*beta(1)*(3*cos(xdata).^2-1))+x(2)*(1+0.5*beta(2)*(3*cos(xdata).^2-1));
```

Appendix.F. Image Generator

```
function varargout = ImageGenerator(varargin)
% Generates Images to input into Process_Data_1_0

% Edit the above text to modify the response to help ImageGenerator

% Last Modified by GUIDE v2.5 30-Apr-2012 12:18:16

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ImageGenerator_OpeningFcn, ...
                   'gui_OutputFcn',  @ImageGenerator_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before ImageGenerator is made visible.
function ImageGenerator_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;guidata(hObject, handles);

%% Set Up Values.
function varargout = ImageGenerator_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
h=dialog('Position',[650 570 120 60]);
uicontrol(h,'Style','text','String','Loading...'),figure(h)
handles.value.peakbeta(1) = 1;
handles.value.peakamp(1)  = 1;
handles.value.peakwidth(1)= 1;
handles.value.linewidth   = 0.5;
handles.value.picnoise    = 0;
set(handles.peakbeta,  'String', handles.value.peakbeta(1) )
set(handles.peakamp,   'String', handles.value.peakamp(1)  )
set(handles.peakwidth, 'String', handles.value.peakwidth(1))
set(handles.linewidth, 'String', handles.value.linewidth )
set(handles.picnoise,  'String', handles.value.picnoise  )
load ref
% [ref1 ref2]=refgen(250);% USE THIS IF YOU WANT NEW PIC SIZE
handles.value.imsize      = size(ref1,1)*2;
handles.value.peakrad(1)  = round(handles.value.imsize/4);
set(handles.peakrad,   'String', handles.value.peakrad(1))
handles.value.ref1        = ref1;
handles.value.ref2        = ref2;
guidata(hObject, handles);
datagenerate(hObject,handles);
close(h)

%create a reference tensor where all the radeii and angles are stored
function [ref1 ref2]=refgen(size)
%note: this function is not used by anything, but is stored here in case
%you want to change the default pic size (use code commented above)
ref1=zeros(size,size,size);ref2=ref1;b=0;
for x=1:size
    b=b+1;h=waitbar(b/size);
    for y=1:size
        for z=1:size
            ref1(x,y,z)=sqrt((x)^2+(y)^2+(z)^2);%calculate radius
```

```
            a=atan(sqrt((y)^2+(z)^2)/(x));      %calculate angle from x axis
            ref2(x,y,z)=(3*(cos(a))^2)-1;                %store (do some processing now to save time later)
        end
    end
end
close(h)
ref2(isnan(ref2))=0;
save ref ref1 ref2

%% Export Button
function ExpData_Callback(hObject, eventdata, handles)
% global Data
[Norm path] = getpath;
%make variables
c       = handles.value.imsize/2; %image width
a       = get(handles.rselect,'String');%string array from dropdown list of datasets
if ~iscell(a);a={a};end
cg      = 0;                    %reset cumulative gaussian
dd      = 0;                    %reset delays(i-1)
shotspp = 1;
shotsbg = 1;
scannum = 1;
if get(handles.sliceon,'Value'), Data = handles.value.sData;
else, Data = handles.value.Data;  end
mx      = max(max(max(Data)));

%create input dialog
titles  = cat(2,{'How many pictures per scan would you like?','filename:',...
    'Cross Correlation (fs):'},strcat('Decay Time Constant (fs):',a'));%titles for editboxes
b       = strcat(a','000');b(end)={'Inf'}; %i.e. {'1000','2000','Inf'}
values  = cat(2,{'20',date,'200'},b);%default values for editboxes
answer  = inputdlg(titles,'Exporting Pictures',1,values,'on');
name    = char(answer(2));                      %remove the '.mat' from the end
if isempty(answer)%if the user presses cancel
    helpdlg('Not saved this time.','Cancel');
elseif any(isnan(str2double(answer(3:end))))
    errordlg('Decay Constant must be a number or Inf (infinity)','Error');uiwait
    ExpData_Callback(hObject, eventdata, handles)
elseif isnan(str2double(answer(1)))||str2double(answer(1))<=0
    errordlg('Number of pictures must be a positive number','Error');uiwait
    ExpData_Callback(hObject, eventdata, handles)
elseif str2double(answer(1))==1; %export image file only
    data=zeros(c,c);
    for d=1:length(a)           %for each dataset
        data=data+Data(:,:,d); %sum each dataset image*decay to make total image for each decay
    end
    data=data+rand(c)*handles.value.picnoise*mx;%add random noise
    data=[flipud(fliplr(data)) flipud(data); fliplr(data) data];
    c.w=c; c.lr=c.w; c.ud=c.w;
    timestamp = ['Model Data genarated by ''ImageGenerator'' on ' date];
    save([path name '.mat'],'data','timestamp','c')
%export data
else%if everything is OK
    %get current data:
    n       = round(str2double(answer(1)));%get the number of images
    delays  = [linspace(-1000,900,ceil(n/2)) logspace(3,4,floor(n/2))];%create delays
    del     = delays; del(del<0)=0; %positive delays (for exp decay)
    cc      = str2double(answer(3));%cross corelation
    Dataa   = zeros(c*2,c*2,n);         %assign space for Data
    decay   = zeros(n,length(a));   %assign space for exponential decay
    Isize   = [c*2,c*2,n];              %size is saved in file
    Data    = handles.value.Data;   %Get the Data
    Datab   = Dataa-128; Datac   = Dataa-128;
    %make new data:
    for i=1:n                       %for each Image
        D=zeros(c,c);
        %calcualte a crude cumulative gaussian with standard deviation cc
        cg  = cg+exp(-(delays(i))^2/(2*cc^2))/(sqrt(2*pi)*cc)*abs((delays(i)-dd));
```

```
        dd  = delays(i);
        for d=1:length(a)           %for each dataset
            tau=str2double(answer(d+3));
            decay(i,d)=(exp(-del(i)/abs(tau))*sign(tau)+(sign(tau)==-1))*cg;
            D=D+Data(:,:,d).*decay(i,d); %sum each dataset image*decay to make total image for each decay
        end
        D=D+rand(c,c)*handles.value.picnoise*mx-128;%add random noise to each image
        Dataa(:,:,i)=[flipud(fliplr(D)) flipud(D); fliplr(D) D];
    end
    Dataa(isnan(Dataa))=-128;
    %save new data
    save([path name 'settings.mat'],'delays','shotspp','shotsbg','scannum','Isize')
    save([path name '-1.mat'],'Dataa','Datab','Datac'),name=[name 'settings'];
    figure,plot(delays,decay),axis tight
end
button=questdlg(['File Saved to ' path name ', Would you like to open this file in Process_Data?'],'Open Exported
File');
if strcmp(button,'Yes')
    close(findall(0,'Tag','ProcessData'))%If Process Data is open, close it
    Process_Data_1_0('file',[path name '.mat']) %Open Process Data with current file
end
%% Add New Data Button
function peakadd_Callback(hObject, eventdata, handles)
a=get(handles.rselect,'String');
if ~iscell(a),a={a};end         %if only one dataset, set up cell array (list of numbers for dropdown list)
c=eval(a{end})+1;               %make new number
a{end+1}=num2str(c);            %add it to end of cell array
%reset to default values for new dataset
handles.value.peakrad(c)   = handles.value.imsize/4;
handles.value.peakbeta(c)  = 1;
handles.value.peakamp(c)   = 1;
handles.value.peakwidth(c) = 1;
set(handles.rselect, 'String',a);
set(handles.rselect, 'Value' ,length(a));
set(handles.peakrad,  'String',handles.value.peakrad(c)  )
set(handles.peakbeta, 'String',handles.value.peakbeta(c) )
set(handles.peakamp,  'String',handles.value.peakamp(c)  )
set(handles.peakwidth,'String',handles.value.peakwidth(c))
datagenerate(hObject,handles);  %generate new data at default values

%% Remove Data Button
function peakdelete_Callback(hObject, eventdata, handles)
a=get(handles.rselect, 'String');
c=get(handles.rselect, 'Value' );
a=a([1:c-1,c+1:end]);
set(handles.rselect, 'String',a);
datagenerate(hObject,handles); %generate data with these values

%% Listbox
function rselect_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function rselect_Callback(hObject, eventdata, handles)
c=get(handles.rselect,'Value');
b=get(handles.rselect, 'String');
a=str2double(char(b(c)));%get the plot number
%set values to the selected dataset's values
set(handles.peakrad,  'String' ,handles.value.peakrad(a)  )
set(handles.peakbeta, 'String' ,handles.value.peakbeta(a) )
set(handles.peakamp,  'String' ,handles.value.peakamp(a)  )
set(handles.peakwidth,'String' ,handles.value.peakwidth(a))
guidata(hObject, handles);

%% Beta Editbox
function peakbeta_CreateFcn(hObject, eventdata, handles)
if                      ispc                        &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function peakbeta_Callback(hObject, eventdata, handles)
```

```
peakbeta=str2double(get(hObject,'String'));%get the new peakbeta value
c=get(handles.rselect,'Value');
b=get(handles.rselect, 'String');
a=str2double(char(b(c)));%get the plot number
if isnan(peakbeta)||peakbeta<-1||peakbeta>2
    set(hObject, 'String', handles.value.peakbeta(a));
    peakbeta = handles.value.peakbeta(a);
    errordlg('Beta Parameter must be between -1 and 2','Error');
end
% Save the new peakbeta value
handles.value.peakbeta(a) = peakbeta; guidata(hObject,handles)
datagenerate(hObject,handles);

%% Radius Editbox
function peakrad_CreateFcn(hObject, eventdata, handles)
if                          ispc                          &&                          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function peakrad_Callback(hObject, eventdata, handles)
peakrad=str2double(get(hObject,'String'));%get the new peakrad value
maxrad=round(handles.value.imsize/2-handles.value.linewidth*2);
c=get(handles.rselect,'Value');
b=get(handles.rselect, 'String');
a=str2double(char(b(c)));%get the plot number
if isnan(peakrad)||peakrad<=0||peakrad>maxrad
    set(hObject, 'String', handles.value.peakrad(a));
    peakrad = handles.value.peakrad(a);
    errordlg(['Input must be a positive number not larger than ' num2str(maxrad)],'Error');
end
% Save the new peakrad value
handles.value.peakrad(a) = peakrad; guidata(hObject,handles)
datagenerate(hObject,handles);

%% Amplitude Editbox
function peakamp_CreateFcn(hObject, eventdata, handles)
if                          ispc                          &&                          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function peakamp_Callback(hObject, eventdata, handles)
peakamp=str2double(get(hObject,'String'));%get the new peakamp value
c=get(handles.rselect,'Value');
b=get(handles.rselect, 'String');
a=str2double(char(b(c)));%get the plot number
if isnan(peakamp)||peakamp<=0
    set(hObject, 'String', handles.value.peakamp(a));
    peakamp = handles.value.peakamp(a);
    errordlg('Amplitude must be a positive number','Error');
end
% Save the new peakamp value
handles.value.peakamp(a) = peakamp; guidata(hObject,handles)
datagenerate(hObject,handles);

%% Peak Width Editbox
function peakwidth_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function peakwidth_Callback(hObject, eventdata, handles)
peakwidth=str2double(get(hObject,'String'));%get the new peakwidth value
c=get(handles.rselect,'Value');
b=get(handles.rselect,'String');
a=str2double(char(b(c)));%get the plot number
if isnan(peakwidth)||peakwidth<0.1
    set(hObject, 'String', handles.value.peakwidth(a));
    peakwidth = handles.value.peakwidth(a);
    errordlg('Peak Width must be a number greater than 0.1','Error');
end
% Save the new peakwidth value
handles.value.peakwidth(a) = peakwidth;
guidata(hObject,handles)
```

```
datagenerate(hObject,handles);

%% Linewidth Editbox
function linewidth_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function linewidth_Callback(hObject, eventdata, handles)
linewidth=str2double(get(hObject,'String'));%get the new linewidth value
if isnan(linewidth)||linewidth<0.1||linewidth>10
    set(hObject, 'String', handles.value.linewidth);
    linewidth = handles.value.linewidth;
    errordlg('Noise Parameter must be a number between 0.1 and 10','Error');
end
% Save the new linewidth value
handles.value.linewidth = linewidth;
b=get(handles.rselect,'String');
v=get(handles.rselect,'Value');
for i=1:length(b) %replot each dataset with new linewidth
    h=waitbar(0);
    %set values
    set(handles.rselect,'Value',i);
    handles.value.Data(:,:,str2double(char(b(i))))=datagenerate(hObject,handles); %generate data, save and plot it
    waitbar(i/length(b))
end
%set the values back to what they should be
set(handles.rselect,'Value',v);
guidata(hObject,handles)
close(h)

%% Noise Editbox
function picnoise_CreateFcn(hObject, eventdata, handles)
if                      ispc                    &&                      isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor')),set(hObject,'BackgroundColor','white');end
function picnoise_Callback(hObject, eventdata, handles)
picnoise=str2double(get(hObject,'String'));%get the new picnoise value
if isnan(picnoise)||picnoise<0||picnoise>2
    set(hObject, 'String', handles.value.picnoise);
    picnoise = handles.value.picnoise;
    errordlg('Noise Parameter must be between 0 and 2','Error');
end
% Save the new picnoise value
handles.value.picnoise = picnoise; guidata(hObject,handles)
datagenerate(hObject,handles);

%% Show Centre Slice
function sliceon_Callback(hObject, eventdata, handles)
if get(hObject,'Value'),set(handles.abelon,'Value',0),end
datagenerate(hObject,handles);

%% Export Graph
function ExpGraph_Callback(hObject, eventdata, handles)
global radius Data beta z
h=findall(0,'Tag','ImageGeneratorGraph'); z=z+1;
if isempty(h), h=figure('Tag','ImageGeneratorGraph'); z=0; end
figure(h), ylabel('Radius (pix)'), xlabel('Image')
plot3(ones(length(radius),1)*z,1:length(radius),radius), axis tight, hold all
if z==0, view(90,0), elseif z==1, view(130,56), rotate3D on, end, axis tight
pubfig
if z==0
    Im = [flipud(fliplr(Data)) flipud(Data); fliplr(Data) Data];
    Im(isnan(Im))=0;
    [rInt beta] = IntegrateImage(Im,1,size(Im,2),180,1,1,1,'Anisotropy Parameters to ');
    figure, beta=permute(beta,[1 3 2]); plot(beta(:,2:3))
end

%% Abel Transform
function abelon_Callback(hObject, eventdata, handles)
if get(hObject,'Value'),set(handles.sliceon,'Value',0),end
```

```
datagenerate(hObject,handles);

%Abel Transform Data
function Data=abelt(Data)
w = size(Data,2);
A = AbelA(w); A1=inv(A);                                    %create the inverse Area Matrix
Data=permute(Data,[2 1 3]);%display images sideways
for ll=1:size(Data,3)
    Data(:,:,ll) = 0.5*A1*Data(:,:,ll);
end
Data=permute(Data,[2 1 3]);
%% Generate Data
function Data2D=datagenerate(hObject,handles)
global Data radius beta
load MyColormaps/ProcessDefault.mat
ff    = get(handles.rselect,'Value');
bb    = get(handles.rselect, 'String');
try a = str2double(char(bb(ff)));   %get the plot number
catch set(handles.rselect,'Value',1); a=1; end
rad   = handles.value.peakrad(a);   %radius of sphere
w     = handles.value.linewidth+handles.value.peakwidth(a);%focus (1=very sharp, 10=fuzzy)
b     = handles.value.peakbeta(a);  %anisotropy parameter
ref1  = handles.value.ref1;         %tensor of radei at each point in 3D space
ref2  = handles.value.ref2;         %tensor of (3*cos(angle)^2-1) at each point in 3D space
c     = handles.value.imsize;       %width of 2D image
slice = get(handles.sliceon,'Value');
%calculate sphere:
f=exp(-((ref1-rad)/w).^2)./(ref1.^2);%radial dependance
g=(1+0.5*b*ref2);                   %angular dependance
Data3D=f.*g;                        %total a-r data
Data3D(~isfinite(Data3D))=0;
% Data3D(isinf(Data3D))=0;

%Squash Sphere to 2D
Data2D=sum(Data3D,3); %squash into 2D image and scale

%save values
handles.value.sData(:,:,a)= Data3D(:,:,1);
handles.value.Data(:,:,a) = Data2D;
handles.value.peakrad(a)  = rad;
guidata(hObject,handles)
Data=zeros(c/2);
for i=1:length(bb) %add all the different datasets together
    a=str2double(char(bb(i)));%get the plot number
    if slice, Data = Data+handles.value.sData(:,:,a)* handles.value.peakamp(a);
    else      Data = Data+handles.value.Data(:,:,a)*  handles.value.peakamp(a); end
end
mx=max(max(Data));
if ~slice, Data = Data+rand(c/2)*handles.value.picnoise*mx; end %add random noise

% ben='lsqlin';% ben=[],'inv','lsqnonneg','lsqlin'
ben=[];
if get(handles.abelon,'Value')
    if ~isempty(ben),
        Im = [flipud(fliplr(Data)) flipud(Data); fliplr(Data) Data];
        [A B Im] = VMIabelinv2(Im, ben);
    else,
        Data = abelt(Data); Im = [flipud(fliplr(Data)) flipud(Data); fliplr(Data) Data];
    end
else Im = [flipud(fliplr(Data)) flipud(Data); fliplr(Data) Data];
end

imshow(Im,[0 Inf],'Parent',handles.axes1),colormap(cmp)
[rInt beta radius] = IntegrateImage(Im,1,c,180,1,0,1,'no');
if slice, mx=max(max(radius))/800; end
radius = radius/mx/800;
plot(handles.axes2,radius), axis tight
```

Appendix.G. BKW Tunnelling Calculations

```
function t=BKWTunneling(v1,v2,uu,e,v_OH,name)
%default values
if nargin<6, name = 'Phenol';       %name
if nargin<5, v_OH = 3582;           %O-H stretch Freq in cm-1
if nargin<4, e    = 6.75;           %H atom energy in eV
%PE surface X vals in Angstroms
if nargin<3, uu   = [0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5];
%PE surface 2 Y vals in eV
if nargin<2, v2   = [9.92 8.50 7.95 7.56 7.24 6.99 6.79 6.67 6.62];
%PE surface 1 Y vals in eV
if nargin<1, v1   = [7.72 6.35 6.09 6.39 6.96 7.65 8.37 9.09 9.76];
end,end,end,end,end,end
%if getting called from 'Caluculate Again' button
if ischar(v1)
    if strcmp(get(get(0,'CurrentFigure'),'Tag'),'BKW')
        %get previous user values
        vals = get(gcf,'UserData');
        name = vals{1};
        v1   = vals{2};
        v2   = vals{3};
        uu   = vals{4};
        e    = vals{5};
        v_OH = vals{6};
    else v1=[]; end
end
%get user's values
prompt={sprintf(['\nPlease Enter the Following Values and Press OK to Calculate\n\n\n' ...
    'Name']),...
    'Potential Energy Surface 1 - Y Values (in eV)',...
    'Potential Energy Surface 2 - Y Values (in eV)    (leave empty if barrier is only on one surface)',...
    'Potential Energy Surface - X Values (in Angstroms)',...
    'Wavepacket Energy (in eV)',...
    'Bond Stretch Frequency (cm-1)'};
defAns = {name,mat2str(v1),mat2str(v2),mat2str(uu),num2str(e),num2str(v_OH)};
titl='BKW Tunneling Rate Calculation';
vals = inputdlg(prompt,titl,1,defAns);
if isempty(vals),return,end     %if user presses cancel
%make sure there are square brackets around data
for i=2:4;
    if ~strcmp(vals{i}(1),'['),vals{i}(2:end+1)=vals{i};vals{i}(1)='[';end
    if ~strcmp(vals{i}(end),']'),vals{i}(end+1)=']';end
end
%get data from dialog box
name = vals{1};
v1   = eval(vals{2});
v2   = eval(vals{3});
uu   = eval(vals{4});
e    = str2num(vals{5});
v_OH = str2num(vals{6});

%constants
m    = 1.00794*1.66053886E-27; %mass of H in kg
c    = 299793000;
ce   = 1.60217646E-19;
h    = 6.626068E-34;
hbar = h/(2*pi);
units={' centuries';' years';' days';' hours';' mins';' seconds';' ms';' us';' ns';' ps';' fs'}; un='';
mp   =[3.1536e+9,3.1536e+7,8.64e+4,3600,60,1,0.001,1e-6,1e-9,1e-12,1e-15];

%find out where PE surfices overlap (only use first overlap)
if isempty(v1),v1=zeros(size(uu));end
if isempty(v2),v2=zeros(size(uu));end
if any(size(v1)~=size(uu))||any(size(v2)~=size(uu))
    h= errordlg('Please ensure Potential Energy Surface X and Y values are the same length');
    uiwait(h),BKWTunneling; return
end
```

```
v    = zeros(size(uu));
u    = uu;
[x y]= polyxpoly(uu,v1,uu,v2);
if ~isempty(x)
    x= x(1); [bla i] = min(abs(u-x));
    if v1(i-1)>v1(i+1), V=v2; v2=v1; v1=V; end
    v(1:i-1) = v1(1:i-1);
    v(i) = y(1);
    u(i) = x;
    v(i+1:end) = v2(i+1:end);
end
%check where barrier and energy lines cross
[x y]= polyxpoly(u,v,[u(1) u(end)],[e e]);
if length(x)>1      %if they overlap twice or more
    %find the first upward sloping overlap
    for l = 1:length(x)
        [bla i] = min(abs(u-x(l)));
        if v(i-1)<v(i),break,end
    end
    x(1)=x(l);y(1)=y(l);%use it as the first overlap point
    if (l+1)>length(x),x(2)=Inf;else x(2)=x(l+1);end %if there is a overlap after it, use it
    v(u<x(1))=y(1);u(u<x(1))=x(1); %replace the points
    v(u>x(l))=y(2);u(u>x(2))=x(2);
elseif length(x)==1 %if they only overlap once
    if mean(v(u<x))>mean(v(u>x)),v(u>x)=y;u(u>x)=x;
    else v(u<x)=y;u(u<x)=x; end
end
%transform to SI
V_OH = v_OH*c*100;  %in Hz
E    = e*ce;        %in J
V    = v*ce;         %in J
U    = u*1e-10;     %in m

%do calculation
yy=sqrt((V-E));xx=U(yy>0);yy=yy(yy>0);pp=spline(xx,yy);%make variables for simpsons method
area = quad(@(xx)ppval(pp,xx),xx(1),xx(end)); %do the integration (simpsons method)(area under barrier)
%area = trapz(U,sqrt((V-E))); %do the integration (trapezim method)(area under barrier)
t    = 1/(V_OH*exp(-2*sqrt(2*m)/hbar*area)); %get the tunneling time (in seconds)

%display answer
disp(['BKW Tunneling Rate for ' name ' = ' num2str(t) ' seconds'])
for i=1:length(mp)
    tt=t/mp(i);
    if tt>1,un=units{i};break,end
end
f = figure;
uicontrol('Style','pushbutton','String','Calculate  Again','Callback','BKWTunneling(''new'')','Position',[10  10  100
20])
uicontrol('Style','pushbutton','String','?','Callback',@BKWhelp,'Position',[120 10 20 20])
patch([u u(end) u(1)],[v e e],[0.8 0.9 0.9],'EdgeColor','none'),hold on,
plot(uu,v1,uu,v2,[uu(1) uu(end)],[e e]),hold off
% ylim([min(v)*0.99,max(v)*1.03])
title([name ' Tunneling Rate = ' num2str(tt) un])
legend({'Barrier';'PES 1';'PES 2';'Energy'})
xlabel(['H bond length (' char(197) ')'])
ylabel('Potential Energy (eV)')
pubfig(f,[name ' BKW Tunneling Rate'])
set(f,'UserData',{name,v1,v2,uu,e,v_OH,t},'Tag','BKW')

function h=BKWhelp(h,g)
helpdlg(['In order to use this you will need calculated potential energy '...
    'surfaces. If the barrier is made up of one surface, leave one ''Y Values'' field empty. '...
    'Please make sure the X and Y fields contain the same number of elements'])
```

Appendix.H. Make Figures Presentable

```
%Makes a nice Figure with UI to Save as Image or Data
%pubfig takes a figure, makes the graphs presentable, and adds a toolbar.
%The idea is to easily create consistant publishable graphs without having
%to spend lots of time, and also to be usable by those who don't know any
%matlab (great for interfacing with a larger UI for example)
%The main componants of the toolbar are the standard matlab figure toolbar,
%but the 'Save' button has been changed to save figures in a more pretty
%and consistant way, or to save the data on the figure as a txt file. In
%addition, there are 7 extra buttons: to edit the colormap (for 3D plots
%and images); to toggle between black & white and colour; to change the
%font size; to change the line width; and to edit the minor tickmarks.
%There is a context menu that allows the user to copy the figure to the
%clipboard (in windows) and save the figure. The 'copy' option only works
%in windows and sometimes gives an image that looks different to the
%figure. Using 'save' is more robust.
%I hope you enjoy using this :o)
%Example 1: 2D graph with easy calling of pubfig
% x=rand(50,1); x=sort(x);
% y=rand(50,1); y=sort(y);
% figure; plot(x,y,'o')
% hold on, plot(fit(x,y,'poly3'),'r') %only use this line if you have the curve fitting toolbox
% legend({'data points','fitted curve'},'Location','NW')
% xlabel('Time (s)'), ylabel('People')
% pubfig
%Example 2: 3D Graph with full calling of pubfig
% surf(peaks)
% axis tight, shading interp, grid off
% title('Pretty Graph')
% pubfig(gcf, 'C:\My Documents', 'Pretty Graph', 0.5, [], 12, 2, 0, 2, 0)
%Copyright (C) Ruth Livingstone 2012
%Syntax:
%pubfig(fh, path, name, sz, data, FS, LW, BW)
%fh  = figure handle
%path= pathstring where you want to save file
%NOTE:set default path below to easily save pictures in your favorite place
%name= name of figure and default file name
%sz  = size of figure (either: 1 number, size of square figure; 2 numbers:
%[width, height]; or, empty: don't change size) (all between 0 and 1)
%NOTE: if sz is 1, fig will be square, if sz is empty, fig won't change
%data= Data to be saved as csv or txt file if user wants
%NOTE: If data is 1 or not included, savefig will retrieve data from graph
%If you don't want the user to be able to save data, set to [].
%FS  = Font Size (default is 15)
%LW  = Line Width (default is 2)
%BW  = Black and White mode (0 is color, 1 is black/white)
%TF  = title font (default is 3: makes titles, axis labels and legends 3 sizes bigger than axis labels)
function pubfig(fh, name, path,  sz, data, FS, LW, MT, TF, BW)
%Generate default input arguments, if needed
if nargin < 10,BW  = 0;                                 %Default color setting
if nargin < 9, TF  = 3;                                 %Default Title Font Size
if nargin < 8, MT  = [3 3 3];                           %Default Minor Ticks (3 minor ticks between each major tick)
if nargin < 7, LW  = 2;                                 %Default Line Width
if nargin < 6, FS  = 15;                                %Default Font Size
if nargin < 5, data= 1;                                 %set User Data to Auto
if nargin < 4, sz  = [];                                %set Size to not change
if nargin < 3, path= 'C:\Users\DB2.27\Desktop';         %Default file path
if nargin < 2, name= 'PubFig';                          %Default name
if nargin < 1, fh  = gcf;                               %Default figure
end,end,end,end,end,end,end,end,end,end
%Make Figure Size
scr  = get(0,'ScreenSize');                             %make sure sizes are valid
sz(sz<0.1)=0.1; sz(sz>1)=1;
sp  = 100-100*sz;

if length(sz)<2 && ~isempty(sz)                         %if user wants square fig
```

```
    pos=[sp, sp, scr(4)*sz, scr(4)*0.9*sz];             %Make a square fig (regardless of screen shape)
elseif ~isempty(sz)                                     %if figure size set is user defined
    pos=[sp(1), sp(2), scr(3)*sz(1), scr(4)*sz(2)];
else                                                    %if no figure size set
    pos=get(fh,'Position');                             %use current figure size
end

%Automatically get data from graph (if not user defined)
if all(all(data==1)) && ~isempty(data)                  %if no pre-defined user data
    try
        plt = findall(fh, '-property', 'XDataSource');  %get the plots on the figure
        xd  = []; yd = [];  zd = [];                            %make the data variables
        for i = 1:length(plt)                           %for each plot
            x = get(plt(i), 'XData');                   %get the xdata
            y = get(plt(i), 'YData');                   %get the ydata
            z = get(plt(i), 'ZData');                   %get the zdata
            yd(end+1) = Inf; xd(end+1) = Inf; zd(end+1) = Inf;              %set a break (so it's obvious what data
belongs to what)
            xd(end+1 : end+length(x)) = x;              %save the xdata
            yd(end+1 : end+length(y)) = y;              %save the ydata
            zd(end+1 : end+length(z)) = z;              %save the ydata
        end
        try data = [xd;yd;zd]';
        catch data = [xd;yd]'; end                      %make into convenient format (if possible)
    catch, data = []; end                               %if not, don't include data
end

%make the figure title
titl=[name ' Plot. Press ''Save'' to save image'];
if ~isempty(data), titl=[titl ' or data'];end

%Get the required state of the 'Black and White' toggle button
if BW, bwstate = 'on';
else,  bwstate = 'off'; end

%Set up the figure
set(fh,'Name',           titl,...
    'NumberTitle',      'off',...
    'Position',          pos,...
    'PaperPositionMode','auto',...
    'MenuBar',          'none',...
    'Toolbar',          'figure',...
    'UserData',          data,...                       %note: if you want to put data into the workspace
    'Renderer',         'zbuffer',...                   %use - data=get(gcf,'UserData');
    'InvertHardcopy',   'off',...
    'Interruptible',    'on',...
    'DockControls',     'off',...
    'WindowButtonUpFcn',{@format_call,0,0},...
    'Color',            'white');

%edit the toolbar to make it more what we want
load myicons                                            %myicons contains: bw cb fup fdown lup & ldown icons
tb = findall(fh,'Type','uitoolbar');                    %find the toolbar
op = findall(tb,'TooltipString','Open File');           %find the open button
nw = findall(tb,'TooltipString','New Figure');          %find the new file button
pt = findall(tb,'TooltipString','Print Figure');        %find the print button
sv = findall(tb,'TooltipString','Save Figure');         %find the save button
cl = findall(tb,'TooltipString','Insert Colorbar');     %find the colorbar button
lg = findall(tb,'TooltipString','Insert Legend');       %find the legend button
tl = findall(tb,'TooltipString','Show Plot Tools and Dock Figure'); %find the Plot Tools button
ht = findall(tb,'TooltipString','Hide Plot Tools');     %find the Plot Tools button
delete(op,nw,pt)                                        %get rid of irrelivent buttons
set(findall(tb),'HandleVisibility','on')                 %set the handle visibility on so that buttons can be re-
ordered
set(sv,'ClickedCallback',{@save_call,[path '/' name]})  %change the Save button callback
set(cl,'ClickedCallback',{@button_call,'Colorbar'})     %change the Colorbar button callback
set(lg,'ClickedCallback',{@button_call,'Legend'})       %change the Legend button callback
set(tl,'ClickedCallback',{@button_call,'Light'})        %change the Plot Tools button callback
```

```
set(ht,'ClickedCallback', @undock_call)                    %change the Hide Plot Tools button callback
set(ht,'ClickedCallback', @undock_call)                    %change the Hide Plot Tools button callback

%make new buttons (only if pubfig hasn't been called previously on the same figure)
if isempty(findall(tb,'TooltipString','Edit Colormap'))
%make the color editing bottons
    uitoggletool('Parent',          tb,...                 %make Black & White Button
                 'ClickedCallback',  @bw_call,...
                 'TooltipString',    'Turn Black and White',...
                 'CData',            bw,...
                 'Separator',        'on',...
                 'State',            bwstate);
    uipushtool ('Parent',           tb,...                 %make Colormap button
                 'ClickedCallback',  'colormapeditor',...
                 'TooltipString',    'Edit Colormap',...
                 'CData',            cb);
%make the font size and line width buttons
    uipushtool ('Parent',           tb,...                 %make Increase Font Size button
                 'ClickedCallback',  {@format_call,1,0},...
                 'TooltipString',    'Increase Font Size',...
                 'CData',            fup,...
                 'Separator',        'on');
    uipushtool ('Parent',           tb,...                 %make Reduce Font Size button
                 'ClickedCallback',  {@format_call,-1,0},...
                 'TooltipString',    'Reduce Font Size',...
                 'CData',            fdown);
    uipushtool ('Parent',           tb,...                 %make Increase Line Width button
                 'ClickedCallback',  {@format_call,0,0.5},...
                 'TooltipString',    'Increase Line Width',...
                 'CData',            lup);
    uipushtool ('Parent',           tb,...                 %make Reduce Line Width button
                 'ClickedCallback',  {@format_call,0,-0.5},...
                 'TooltipString',    'Reduce Line Width',...
                 'CData',            ldown);
    uipushtool ('Parent',           tb,...                 %make Minor Ticks button
                 'ClickedCallback',  @mTicks_call,...
                 'TooltipString',    'Edit Minor Ticks',...
                 'CData',            mt);
    uipushtool ('Parent',           tb,...                 %make Minor Ticks button
                 'ClickedCallback',  @tick_call,...
                 'TooltipString',    'Edit Major Ticks',...
                 'CData',            Mt);
elseif BW==0, BW=3;
end

%make a context menu
if ispc                                                    %if is running on a windows machine
    fm = uicontextmenu('Parent',fh);                       %make a context menu
    uimenu(fm,  'Label',        'Copy',...                 %make a 'Copy' option to copy to clipboard (windows only)
                'Callback',     {@save_call,0},...         %save it to clipboard (go to save_callback)
                'Accelerator',  'C');                      %why doesn't this work?
    uimenu(fm,  'Label',        'Save',...                 %make a 'save' option (same as save button)
                'Callback',     {@save_call,[path '/' name]},...%save it as a file (go to save_callback)
                'Accelerator',  'S');                      %why doesn't this work?
    ax = findall(fh, 'Type', 'axes','-not', '-property', 'Location');%find all axes that aren't legends (legends have
their own context menus)
    set(ax, 'UIContextMenu', fm);                          %give them the context menu
end

%Make the Graph Pretty
pubgraph(fh,FS,LW,BW,MT,TF)                                %go to pubgraph function

%make sure all the buttons update the figure
buttons = [rotate3d(fh) zoom(fh) pan(fh)];                 %get the handles to the zoom, pan and rotate properties
set(buttons,'ActionPostCallback',{@format_call,0,0})       %make sure they update figure
figure(fh)                                                 %pull the figure to the front
return
```

```
%CALLBACKS REQUIRED BY PUBFIG.M TOOLBAR AND CONTEXT MENU:
%'Save' Button (saves the current figure as an image file, or saves data as a text file)
function h=save_call(h,g,name)
savepretty(gcbf,3,name)                                    %go to savepretty function
return

%Hide Plot Tools button
function h=undock_call(h,g)
set(gcbf,'WindowStyle','normal')                           %undock figure
pubgraph(gcbf,get(gca,'FontSize'),get(gca,'LineWidth'),3,[])%update axes to reflect changes
set(gcbf,'DockControls','off')                               %turn docking controls back off (they get automatically
turned on by 'Show Plot Tools')
return

%Button Callback
function button_call(h,g,Type)
insertmenufcn(gcbf,Type)                                   %go to the default matlab callback function
if strcmpi(Type,'Light')
    c=findall(gcbf,'Type','light');
    delete(c(1))
    figurepalette('show')
end
pubgraph(gcbf,get(gca,'FontSize'),get(gca,'LineWidth'),3,[])%update axes to refelct changes
return

%'Turn Black and White' Button
function h=bw_call(h,g)
switch get(h,'State')                                      %see if it's pushed up or down
    case 'on', BW=1; case 'off', BW=2;                     %turn that into something pubgraph can understand
end
pubgraph(gcbf,get(gca,'FontSize'),get(gca,'LineWidth'),BW,[])%update axes to refelct changes
return

%Change Font Size or Line Width Buttons
function h=format_call(h,g,f,l)
FS=get(gca,'FontSize')+f;                                  %get fontsize and change it
LW=get(gca,'LineWidth')+l;                                 %get linewidth and change it
pubgraph(gcbf,FS,LW,3,[])                                    %go to pubgraph function (BW=3 means no colors will get
changed or overwritten, and the y axis won't get changed)
return

%Change Minor Ticks
function h=mTicks_call(h,g)
MT = cellstr(num2str(get(findall(gcf,'Tag','mTickAx'),'UserData'))); %get the current Minor Tickmarks Settings
if length(MT)<3, MT = {'0','0','0'}; end                     %check that there are any Minor Tickmarks and reset to
default if there isn't
prompts = {sprintf('How Many Tickmarks Would You Like Between Each Major Tickmark?\n\nX Axis'),'Y Axis','Z Axis'};%get
the text to put above each box
MT = inputdlg(prompts, 'Minor Tickmarks Editor', 1, MT);   %open a dialogue to ask about settings
MT = str2double(cellstr(MT));                              %change the strings into numbers
FS = get(gca,'FontSize');                                  %get the fontsize
LW = get(gca,'LineWidth');                                 %get the linewidth
pubgraph(gcbf,FS,LW,3,MT)                                    %go to pubgraph function (BW=3 means no colors will get
changed or overwritten, and the y axis won't get changed)
return

%Change Major Ticks
function h=tick_call(h,g)
if ischar(h), axis = h;
else, axis = questdlg('On which axis would you like to edit the major tickmark spacing?','','X','Y','Z','X');
end
if isempty(axis), return, end                              %if the user closes window, exit
oldtik  = get(gca,[axis 'Tick']);                          %get the current tickmarks
if length(oldtik)<2, oldtik=eval([axis 'lim'])/5; end
oldspc  = num2str(oldtik(2)-oldtik(1));                    %find out tickmark spacing
spacing = str2double(cellstr(inputdlg('New Spacing',[axis 'Axis Tickmarks'],1,{oldspc}))); %ask user what it wants the
new spacing to be
if isempty(spacing), return                                %if the user presses cancel, exit
```

```
elseif ~isfinite(spacing)
    errordlg('Please enter a number')                       %error dialog
    uiwait, tick_call(axis,g)                               %go back to beginning
    return
end
switch axis
    case 'X',  lim = xlim;
    case 'Y',  lim = ylim;
    case 'Z',  lim = zlim;
    otherwise, lim = xlim; axis='X';
end
tik = floor(lim(1)/spacing)*spacing: spacing :floor(lim(2)/spacing)*spacing; %default to 'spacing' devision tickmarks
set(gca, [axis 'Tick'], tik,[axis 'TickLabel'], num2str(tik'))
pubgraph(gcbf,get(gca,'FontSize'),get(gca,'LineWidth'),3,[])%update axes to reflect changes
return
%% Save graph as a pretty pic or as a data file
%savepretty saves a figure as a nice image or a data file, using a 'save
%as' window to allow the user to save wherever they want, and a range of
%file formats to choose from.
%Example:
% x=rand(50,1); x=sort(x);
% y=rand(50,1); y=sort(y);
% h=figure; plot(x,y,'o')
% xlabel('Time (s)'), ylabel('People')
% savepretty(h,4,'C:\My Documents\')
%Copyright (C) Ruth Livingstone 2012
%syntax:
%savepretty(fh, res, fname)
%fh   = figure handle
%res  = size of final picture (i.e. multiplication factor)
%fname= default file name and path (if fname=0, it means copy to clipboard)
function savepretty(fh, res, fname)
%parts of savepretty are based on print2array.m and export_fig.m (C) Oliver Woodford 2008-2011
if nargin < 3, fname='';                                    %Generate default input arguments, if needed
    if nargin < 2, res = 3;
        if nargin < 1, fh = gcf; end
    end
end
%get the available file formats
savefig={'*.tif','Tiff Image (*.tif)';...
         '*.eps','EPS Image (*.eps)';...
         '*.jpg','JPEG Image (*.jpg)';...
         '*.bmp','BMP Image (*.bmp)'};                      %get the image formats available
Data = get(fh,'UserData');                                  %get the data
if ~isempty(Data);                                          %check if you can save data as well
    savefig=[savefig; {'*.txt;*.csv','data file (*.txt,*.csv)'}]; %add the data formats
end
%open 'save as' window
if fname~=0;                                                %if saving to file
    [FileName,PathName,i] = uiputfile(savefig,'Save figure as',fname); %open a "Save As" box
    fname=[PathName FileName];                              %get new file name
    name=['file saved as: ' FileName];                      %decide what to display on figure
else                                                        %if copying
    name='Image Copied to Clipboard';
    i=10;
end
res_str = ['-r' num2str(ceil(get(0,'ScreenPixelsPerInch')*res))];%Set the resolution parameter
switch i                                                    %choose how to save the data
    case 0, return                                          %if user presses cancel
    case 1, ext='tif';
    case 2, ext='eps';
    case 3, ext='jpg';
    case 4, ext='bmp';
    case 5, ext='csv';
    case 10,ext='';
end
%make the image the right way round
orient = get(fh, 'PaperOrientation');
```

```
if i~=2, set(fh, 'PaperOrientation', 'portrait'), end
%stop minor tickmark axis and normal axis becoming misaligned in the printing process
a  = findall(fh,'ActivePositionProperty','OuterPosition');
sz = get(a,'Position');
if ~iscell(sz), sz={sz}; end
set(a,'ActivePositionProperty','Position');
%print image to a temporory file
eon= isempty(strfind(fname,'.'));               %check if the extension is included
if eon, fname=[fname '.' ext]; end               %add the extension if required
tmp_nam = [tempname '.tif'];                    %generate temporory file name
print(fh,'-zbuffer',res_str,'-dtiff',tmp_nam,'-noui');  %Print to temporary tiff file
A = imread(tmp_nam);                            %Read the printed file
delete(tmp_nam);                                %Delete the temporary file
%crop the background                            (adapted from export_fig.m - 'crop_background' fn)
[h w c] = size(A); bc=A(1,1,:);                 %#ok<NASGU>
for l = 1:w
    if cropb(A(:,l,:),bc),break,end
end
for r = w:-1:l
    if cropb(A(:,r,:),bc),break,end
end
for t = 1:h
    if cropb(A(t,:,:),bc),break,end
end
for b = h:-1:t
    if cropb(A(b,:,:),bc),break,end
end
A = A(t:b,l:r,:);
%save the image
switch i
    case 2, print(fh,'-painters',res_str,'-depsc2',fname,'-noui');%print to eps (painters is required to print as
vector)
    case 5, if iscell(Data);Data=cell2mat(Data);end,csvwrite(fname,Data), open(fname) %write to txt file
    case 10, print(fh,'-painters',res_str,'-dmeta','-noui');    %print to clipboard
    otherwise, imwrite(A,fname,ext)                    %save as image file
end
set(fh,'Name', name, 'PaperOrientation', orient)        %change figure title
for i=1:length(a)
    set(a(i),'Position',sz{i},'ActivePositionProperty','OuterPosition');%the axes tended to move when I did this, so
put them back.
end
% set(a,'ActivePositionProperty','OuterPosition');        %put back to normal
pubgraph(gcbf,get(gca,'FontSize'),get(gca,'LineWidth'),3,[])%redraw graph
return

%crop background fn
function exit=cropb(A,col)
%(returns true if 'A' contains any color other than col)
exit = 0;
for a = 1:size(A,3)
    if ~all(A(:,:,a) == col(a))
        exit = 1;
        break;
    end
end
return

%% Publish graph function (makes graphs pretty and publishable)
%pubgraph takes a figure and makes it pretty and publishable. The font
%size, line width, background color, and line colors can be set, and all
%the axes labels are set to the same format.
%Example 1:
% x=rand(50,1); x=sort(x);
% y=rand(50,1); y=sort(y);
% h=figure; plot(x,y,'o')
% xlabel('Time (s)'), ylabel('People')
% pubgraph(h,15,2,0,[3 3 3],2)
%Example 2: create a simple ui, and improve appearence
```

```
% x=rand(50,1); x=sort(x);
% y=rand(50,1); y=sort(y);
% h=figure;
% subplot 211, plot(x,y,'o')
% subplot 212, surf(peaks)
% xlabel('Time (s)'), ylabel('People')
% uicontrol(h,'Style','pushbutton',...
%     'String','Refresh',...
%     'Callback','subplot 211, x=sort(rand(50,1));y=sort(rand(50,1));plot(x,y,''o''),pubgraph(gcbf,13,2)')
% pubgraph(h,13,2,0,[],2)
%Copyright (C) Ruth Livingstone 2012
%syntax:
%pubgraph(fh,FS,LW,BW,MT)
%fh = handles to the figure containing the graph
%FS = Font Size
%LW = Line Width
%BW = {0,1,2} toggles black and white status of graph
%MT = minor ticks (e.g. [0 2] gives no X ticks and 2 minor ticks between each major tick)
%TF = title font (e.g. 2 makes titles, axis labels and legends 2 sizes bigger than axis labels)
%(0 changes nothing, 1 turns graph black and white, 2 turns a black and
%white graph back into color. 2 should only be called after 0 or 1)
function pubgraph(fh,FS,LW,BW,MT,TF)
persistent lcol cmp
%get current font sizes
if nargin<1, fh = gcf; end
a=get(gca,'FontSize');
t=get(findall(fh,'Type','text','-not','String',''),'FontSize');
figure(fh)
if isempty(t), t={a}; elseif ~iscell(t), t={t}; end
if nargin<6, TF = t{end}-a;                         %get current titles vs axes font scale
    if nargin<5, MT = [];                            %default minor ticks (don't add any, but if they're there, don't
remove them)
        if nargin<4, BW = 0;                        %default black&white setting (black&white off)
            if nargin<3, LW = get(gca,'LineWidth');  %default line width (thickish lines)
                if nargin<2, FS = get(gca,'FontSize');                %default font size (size 15)
                end
            end
        end
    end
end

% set(fh,'Renderer','zbuffer')
% figure(fh)     %pull the figure forwards
if isempty(MT), MT = get(findall(gcf,'Tag','mTickAx'),'UserData'); end
if iscell(MT),  MT = MT{1}; end
delete(findall(fh,'Tag','mTickAx'))          %delete old minor ticks
if any(MT)
    if length(MT)<2,    MT(2)=MT(1);
        if length(MT)<3,MT(3)=MT(1); end
    end
end
MT = MT(:);                                         %make sure it's a column vector;
FS(FS<1)=1; LW(LW<0.5)=0.5;                         %make sure the input values are valid
set(findall(fh, 'Type', 'axes'), 'FontSize', FS, 'LineWidth', LW)%make everything on the axis correct,
lines= findall(fh, 'Type', 'line');
set(lines, 'LineWidth', LW);                        %make all lines (apart from the axes lines) correct
set(findall(fh,  'Type',  'text'),  'FontSize',  FS+TF);%make  all  text    (apart  from  the  axes  lines)  correct
,'FontName','Calibri'
if BW<2
    lcol= [num2cell(lines(:)) get(lines,'Color') get(lines,'MarkerFaceColor')];%save the old line colors
    cmp = colormap;                                 %save the old colormap
end
if BW==1                                            %if turn the figure black and white
    set(lines, 'Color','black','MarkerFaceColor','black'); %set all lines black
    colormap gray                                   %set the colormap grayscale
elseif BW==2;                                       %if turn the figure back into colors
    colormap(cmp)                                   %put the colormap back
    for i=1:size(lcol,1)                            %for each plot line
```

```
        if ishandle(lcol{i,1})                    %check that the line still exists
            set(lcol{i,1},'Color',lcol{i,2},'MarkerFaceColor',lcol{i,3}); %put it back to it's old color
        end
    end
end
drawnow

%get axes handles
axs   = findall(fh, 'Type', 'axes','-not', '-property', 'Location'); %get all the axes apart from  legends and
colorbars
leg  = findall(fh, 'Type', 'axes', '-property', 'Location'); %get all the legends
h    = rotate3d(fh);                                  %get the handles to the rotate3d properties
%for each graph (but not legend or colorbar) on the figure
for i= 1:length(axs)
%make this axis the curent axis
    ax     = axs(i);
    axes(ax)
%make sure x and y axes aren't too tight
    [az el]= view;
    twoD   = (diff([az el])==90);                   %find out if axis is 2D or 3D
    v      = ylim;                                  %get current y axis limits
    sc     = 0.05*diff(v);                          %add in 5% each side
    zer    = abs(v(1)/v(2))<0.02;                   %see if axis begins near zero
    if ~BW && twoD && ~zer                           %don't do for BW, since then graph will keep extending everytime
user hits toggletool
        ylim([v(1)-sc , v(2)+sc]),                  %apply it to the graphs
    end
    v      = xlim;                                  %get current x axis limits
    sc     = 0.02*diff(v);                          %add in 2% each side
    zer    = abs(v(1)/v(2))<0.02;                   %see if axis begins near zero
    if ~BW && twoD && ~zer                          %apply it to the graphs (don't do it if the axis begins at zero)
        xlim([v(1)-sc , v(2)+sc]),
    end

%format X and Y axis tickmarks so all numbers are displayed with the same precision
    YTick = char(get(ax,'YTickLabel'));             %get the current Y axis labels
    if ischar(YTick),YTick = str2num(YTick);end
    if ~isempty(YTick), YTickL=num2str(YTick(:),tickfmt(YTick));
    else YTickL=get(ax,'YTickLabel'); end
    XTick = char(get(ax,'XTickLabel'));             %get the current X axis labels
    if ischar(XTick),XTick = str2num(XTick);end
    if ~isempty(XTick), XTickL=num2str(XTick(:),tickfmt(XTick));
    else XTickL=get(ax,'XTickLabel'); end
    set(ax,'YTick',      get(ax,'YTick'),...        %set all the y labels to the same precision
           'YTickLabel', YTickL,...
           'XTick',      get(ax,'XTick'),...
           'XTickLabel', XTickL);
   if ~isempty(findall(ax,'-property','ZTick'))     %if there is a Z axis
        ZTick = str2num(char(get(ax,'ZTickLabel'))); %get the current Z axis labels
        if ~isempty(ZTick), ZTickL=num2str(ZTick(:),tickfmt(ZTick));
        else ZTickL=get(ax,'ZTickLabel'); end
        set(ax,'ZTick',get(ax,'ZTick'),...          %set all the y labels to the same precision
           'ZTickLabel',ZTickL);
   end

%minor tickmarks: draw new axes underneath current axes
   if any(MT) && strcmp(get(ax,'Visible'),'on')     %check if we want to plot minor tickmarks
       a2=axes('Parent',        fh,...
               'Position',      get(ax, 'Position'),... %make new axis in exactly the same place as old axes
               'LineWidth',     LW,...
               'FontSize',      FS,...
               'Color',         'white',...
               'XTick',         getTick(get(ax,'XTick'),MT(1)),...  %get new tick locations...
               'YTick',         getTick(get(ax,'YTick'),MT(2)),...  %using getTick function (below)
               'ZTick',         getTick(get(ax,'ZTick'),MT(3)),...
               'XTickLabel',    '',...              %don't label minor tickmarks
               'YTickLabel',    '',...
               'ZTickLabel',    '',...
```

```
                'XLim',          get(ax,'XLim'),...  %set the axes limits to the same as the old axes
                'YLim',          get(ax,'YLim'),...
                'ZLim',          get(ax,'ZLim'),...
                'Box',           get(ax,'Box') ,...  %set the Box properties to the same as the old axes
                'View',          get(ax,'View'),...
                'TickLength',    [.007 .014],...     %set the ticklength to be short
                'Tag',           'mTickAx',...       %Tag it (so we can find it again easily)
                'UserData',      MT,...              %Save the minor tick settings for next time
                'Layer',         'Bottom');
        set(ax, 'TickLength',    [.014 .028],...     %make the major tickmarks a bit bigger
                'Color',           'none');              %make the axes transparent (so you can see the tickmark axes
behind it)
        setAllowAxesRotate(h,a2,false);              %don't allow the user to rotate the background axes
    end
    axes(ax)                                         %make original axes current (for rotating etc)
end
if ~isempty(leg), axes(leg), end                     %bring legend to the front (so it's over any lines)
return
%get minor tick mark positions
function tick=getTick(TICK,MT)
%TICK = major tick mark positions (vector)
%MT   = no. of minor tickmarks per devision (number)
%tick = minor tick mark positions (vector)
tick=[];
if isempty(TICK) || MT<=0; %if no minor ticks
    return
end
%create tickmarks beyond the confines of the graph
TICK(2:end+1) = TICK;                                %make space for new values
TICK(1)       = TICK(2)-(TICK(3)-TICK(2));           %make new min value
TICK(end+1)   = TICK(end)+(TICK(end-1)-TICK(end-2));%make new max value
%create minor tickmarks vector
for i=2:length(TICK)                                 %for each major devision
    sp = (TICK(i)-TICK(i-1))/(MT+1);                 %get the minor tickmark spacing
    ti = TICK(i-1):sp:TICK(i);                       %get the range of the current major devision
    tick = [tick ti(2:end-1)];                       %create the minor tickmark positions
end
return
%return the best (consitant) format for a list of numbers
function fmt = tickfmt(Tick)
%fmt = best format
%Tick= list of numbers
if ~any(isrem(Tick,1))
    fmt='%-5.0f';
elseif ~any(isrem(Tick,0.1))
    fmt='%-5.1f';
elseif ~any(isrem(Tick,0.01))
    fmt='%-5.2f';
elseif ~any(isrem(Tick,0.001))
    fmt='%-5.3f';
else
    fmt='%-5.4f';
end
return
%check if there are any remainders
function c = isrem(a,b)
%a= quotient (number or matrix of numbers)
%b= divisor  (number or matrix the same size as a)
%c= same size as a. 0 if no remainders, 1 if remainder.
rems = (round(a./b)-a./b).*b;                        %get remainders (same as rem(a,b))
c = abs(rems) > 1e-9;                                    %check if they exist (>1e-9 rather than ~=0 because of matlab
rounding errors)
return
```